Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Системы обработки информации и управления»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОМУ ПРОЕКТУ

НА ТЕМУ:

«Решение задачи регрессии»

Студент ИУ5-62Б _____ **В.О. Шушпанов**
(Группа) (Подпись, дата) (И.О.Фамилия)

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

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

2020 г.

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ
Заведующий кафедрой ____ИУ5___
(Индекс)
_____ В.М. Черненький_
(И.О.Фамилия)
« _____ » _____ 20 ____ г.

З А Д А Н И Е
на выполнение курсового проекта

по дисциплине __«Технологии машинного обучения»_____

Студент группы ИУ5-62Б

_____Шушпанов Владислав Олегович_____
(Фамилия, имя, отчество)

Тема курсового проекта «Решение задачи регрессии»_____
_____
_____

Направленность КП (учебный, исследовательский, практический, производственный, др.)
_____учебный_____
Источник тематики (кафедра, предприятие, НИР) _____кафедра_____

График выполнения проекта: 25% к 3 нед., 50% к 9 нед., 75% к 12 нед., 100% к 16 нед.

Задание Решение задачи машинного обучения. Результатом курсового проекта является отчет, содержащий описания моделей, тексты программ и результаты экспериментов._____
_____
_____
_____
_____
_____

Оформление курсового проекта:

Расчетно-пояснительная записка на 34 листах формата А4.
Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)
_____
_____
_____
_____

Дата выдачи задания « 7 » февраля 2020 г.

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

Студент _____ **В.О. Шушпанов**
(Подпись, дата) (И.О.Фамилия)

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

Содержание

Введение

Курсовой проект – самостоятельная часть учебной дисциплины «Технологии машинного обучения» – учебная и практическая исследовательская студенческая работа, направленная на решение комплексной задачи машинного обучения. Результатом курсового проекта является отчет, содержащий описания моделей, тексты программ и результаты экспериментов.

Курсовой проект опирается на знания, умения и владения, полученные в рамках лекций и лабораторных работ по дисциплине.

В рамках курсового проекта было проведено типовое исследование – решение задачи машинного обучения на основе материалов дисциплины.

Основная часть

Задание

Схема типового исследования, проводимого студентом в рамках курсовой работы, содержит выполнение следующих шагов:

1. 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.
2. Проведение разведочного анализа данных. Построение графиков, необходимых для понимания структуры данных. Анализ и заполнение пропусков в данных.
3. Выбор признаков, подходящих для построения моделей. Кодирование категориальных признаков. Масштабирование данных. Формирование вспомогательных признаков, улучшающих качество моделей.
4. 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.
5. Выбор метрик для последующей оценки качества моделей. Необходимо выбрать не менее трех метрик и обосновать выбор.
6. Выбор наиболее подходящих моделей для решения задачи классификации или регрессии. Необходимо использовать не менее пяти моделей, две из которых должны быть ансамблевыми.
7. Формирование обучающей и тестовой выборок на основе исходного набора данных.
8. 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.
9. 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.
10. Повторение пункта 8 для найденных оптимальных значений гиперпараметров. Сравнение качества полученных моделей с качеством baseline-моделей.
11. 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.

Приведенная схема исследования является рекомендуемой. В зависимости от решаемой задачи возможны модификации.

Последовательность действий

1) Поиск и выбор набора данных для построения моделей машинного обучения.

В работе используется набор данных, состоящий из 10 колонок и 53939 наблюдений. 7 столбцов – это числовые атрибуты алмаза. Этот набор данных предназначен для определения цены, с помощью собранных данных о алмазах. Набор данных содержит следующие колонки:

1. цена в долларах США (price)
2. вес алмаза (carat)
3. качество среза (cut)
4. цвет алмаза (color)
5. показатель чистоты алмаза (clarity)
6. общая глубина в процентах (depth)
7. ширина сверху алмаза относительно самой широкой точки (table)
8. длина в мм (x)
9. ширина в мм (y)
10. глубина в мм (z)

In [1]:

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

In [2]:

```
col_list = ['carat',
                'cut',
                'color',
                'clarity',
                'depth',
                'table',
                'price',
                'x',
                'y',
                'z']
data = pd.read_csv('diamonds.csv', names=col_list, header=1, sep=",")
```

2) Проведение разведочного анализа данных. Построение графиков, необходимых для понимания структуры данных.

In [3]:

```
data.head()
```

Out[3]:

	carat	cut	color	clarity	depth	table	price	x	y	z
2	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
3	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
4	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
5	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75
6	0.24	Very Good	J	VVS2	62.8	57.0	336	3.94	3.96	2.48

In [4]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 53939 entries, 2 to 53940
Data columns (total 10 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   carat    53939 non-null  float64
 1   cut      53939 non-null  object 
 2   color    53939 non-null  object 
 3   clarity  53939 non-null  object 
 4   depth    53939 non-null  float64
 5   table    53939 non-null  float64
 6   price    53939 non-null  int64  
 7   x        53939 non-null  float64
 8   y        53939 non-null  float64
 9   z        53939 non-null  float64
dtypes: float64(6), int64(1), object(3)
memory usage: 4.5+ MB
```

In [5]:

```
data = data[:10000]
```

In [6]:

```
data.shape
```

Out[6]:

```
(10000, 10)
```

In [7]:

```
data.columns
```

Out[7]:

```
Index(['carat', 'cut', 'color', 'clarity', 'depth', 'table', 'price', 'x', 'y',
       'z'],
      dtype='object')
```

In [8]:

```
data.dtypes
```

Out[8]:

```
carat      float64
cut         object
color       object
clarity     object
depth      float64
table      float64
price        int64
x          float64
y          float64
z          float64
dtype: object
```

In [9]:

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

Out[9]:

```
carat      0
cut        0
color      0
clarity    0
depth      0
table      0
price      0
x          0
y          0
z          0
dtype: int64
```

In [10]:

```
def code_mean(data, cat_feature, real_feature) :
    return (data[cat_feature].map(data.groupby(cat_feature)[real_feature].mean()))
```

In [11]:

```
data['cut_mean_price'] = code_mean(data, 'cut', 'price')
data['color_mean_price'] = code_mean(data, 'color', 'price')
data['clarity_mean_price'] = code_mean(data, 'clarity', 'price')
```

In [12]:

```
del data['cut']
del data['color']
del data['clarity']
```

In [13]:

```
data.head()
```

Out[13]:

	carat	depth	table	price	x	y	z	cut_mean_price	color_mean_price	clarity_mean_price
2	0.21	59.8	61.0	326	3.89	3.84	2.31	3503.800317	3338.471823	3430.389793
3	0.23	56.9	65.0	327	4.05	4.07	2.31	3541.230008	3338.471823	3105.061500
4	0.29	62.4	58.0	334	4.20	4.23	2.63	3503.800317	3286.895131	3141.847995
5	0.31	63.3	58.0	335	4.34	4.35	2.75	3541.230008	3589.101161	3790.572568
6	0.24	62.8	57.0	336	3.94	3.96	2.48	3414.363204	3589.101161	2912.161850

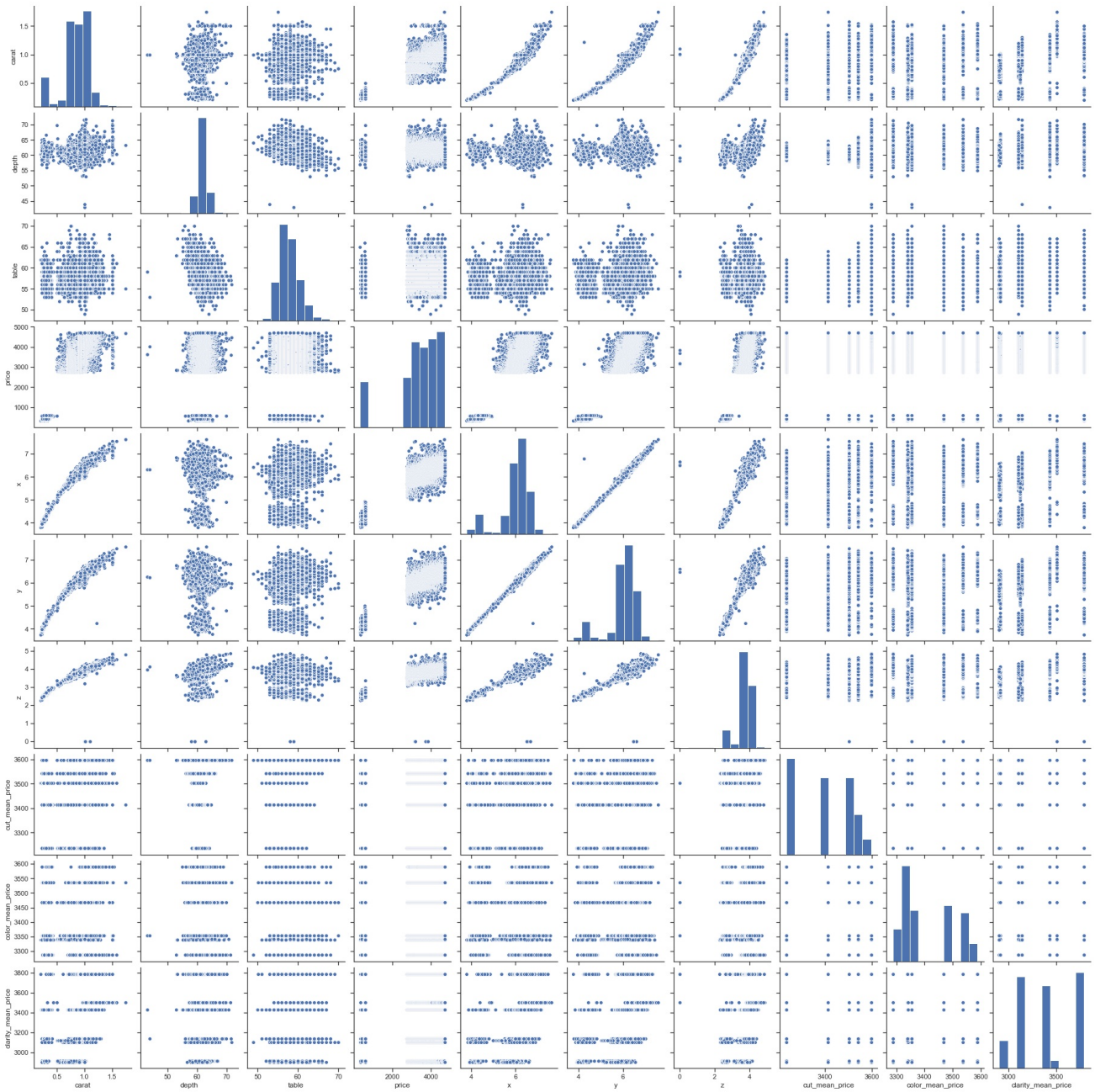
Набор данных не содержит пропусков, категориальные пизнаки закодированы.

In [14]:

```
sns.pairplot(data)
```

Out[14]:

```
<seaborn.axisgrid.PairGrid at 0x1a276bbfd0>
```

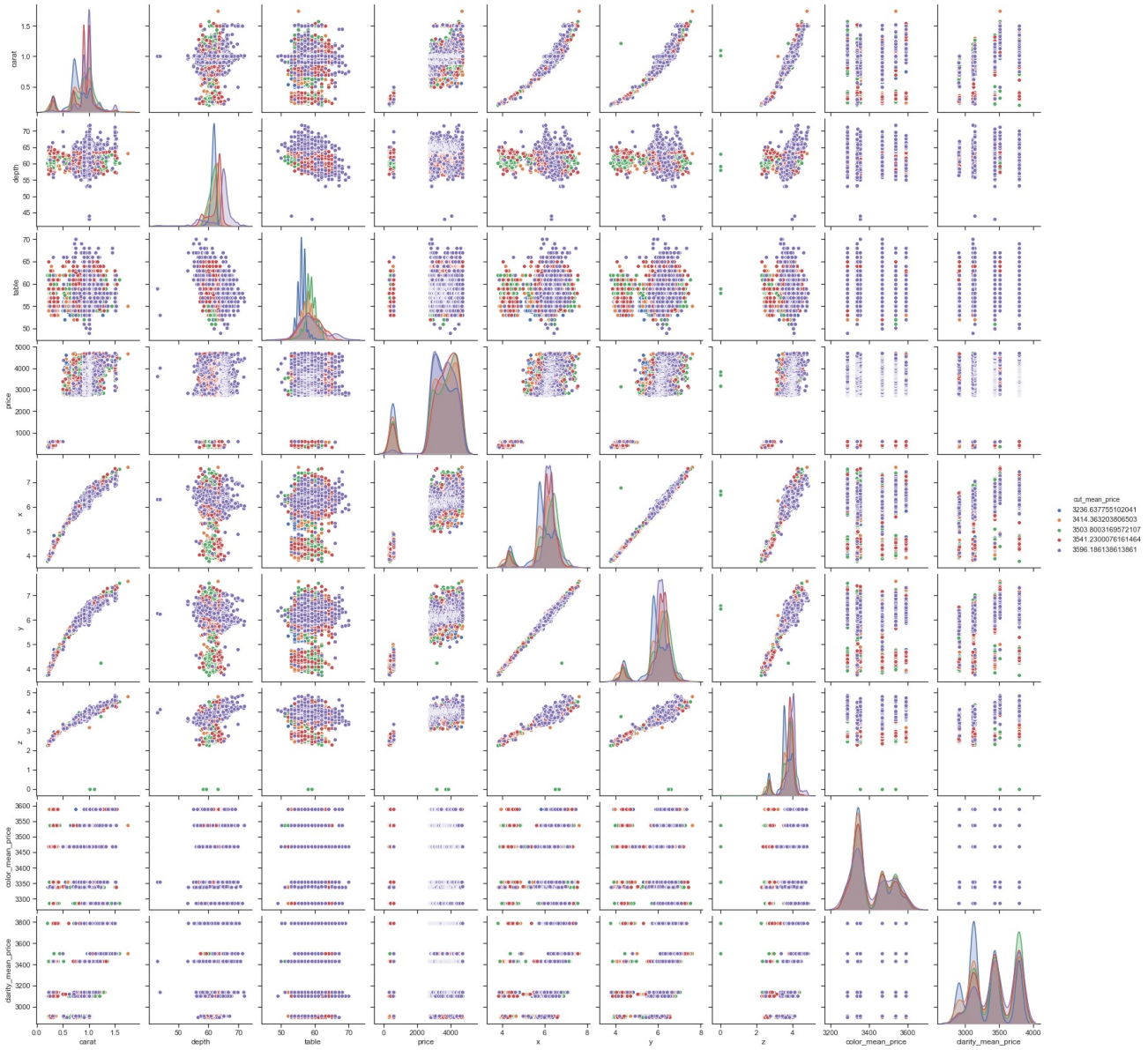

carat
depth
table
price
x
y
z
cut_mean_price
color_mean_price
clarity_mean_price

In [15]:

```
sns.pairplot(data, hue="cut_mean_price")
```

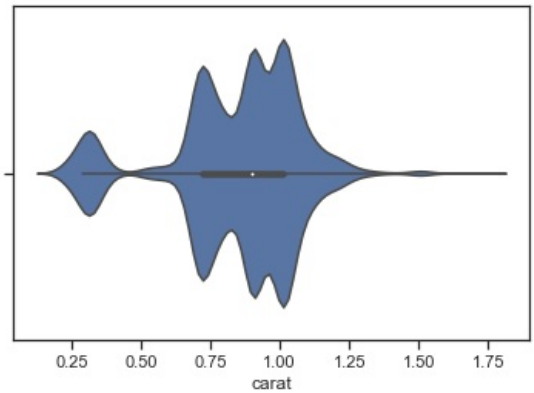
Out[15]:

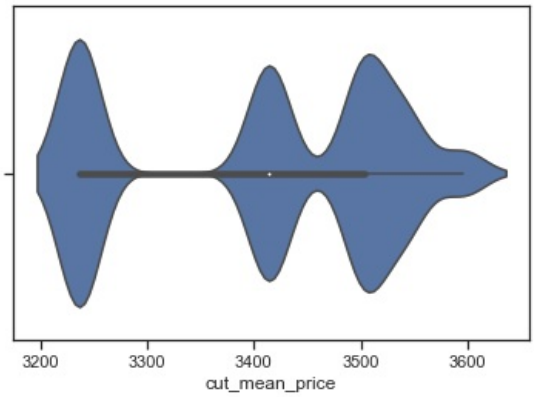
```
<seaborn.axisgrid.PairGrid at 0x10fb28350>
```

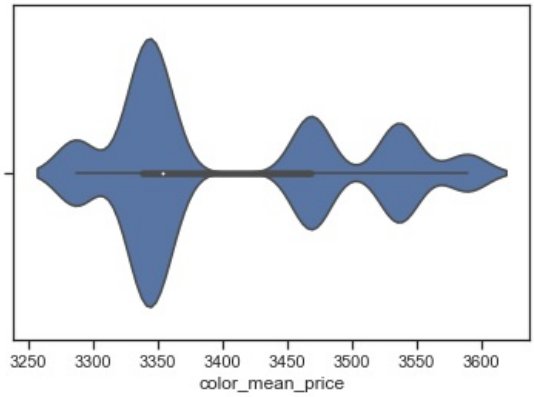


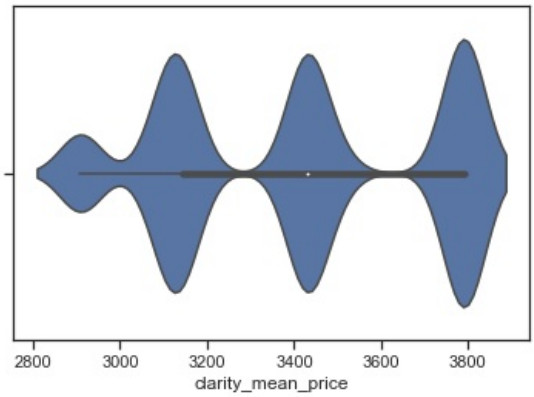
In [16]:

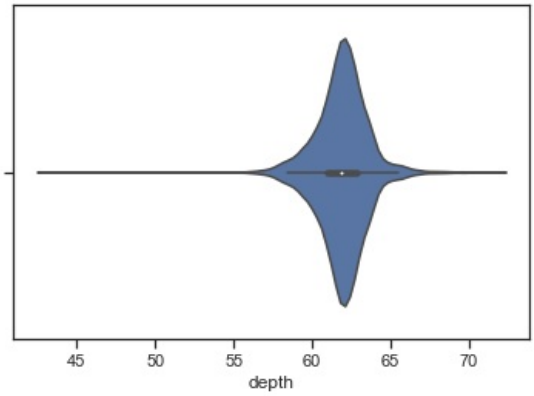
```
# Скрипичные диаграммы для числовых колонок
for col in ['carat',
                'cut_mean_price',
                'color_mean_price',
                'clarity_mean_price',
                'depth',
                'table',
                'price',
                'x',
                'y',
                'z']:
    sns.violinplot(x=data[col])
    plt.show()
```

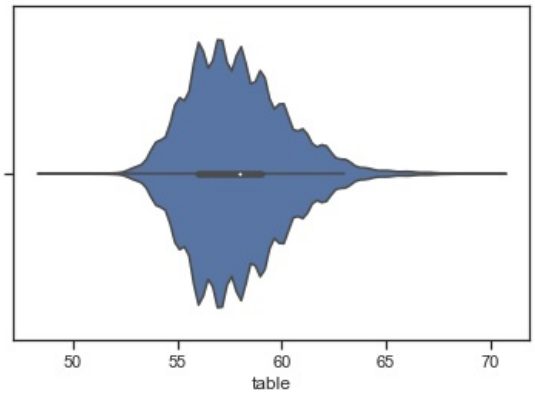

0.25
0.50
0.75
1.00
1.25
1.50
1.75
carat

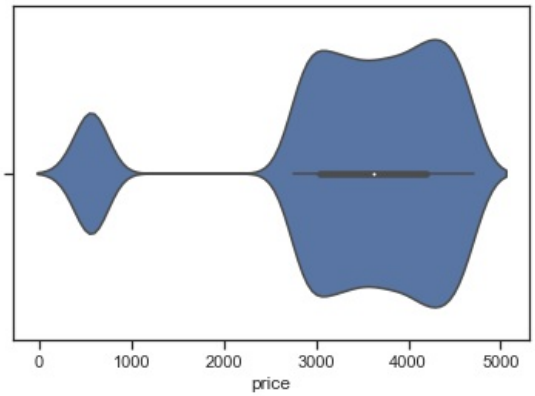

3200
3300
3400
3500
3600
cut_mean_price

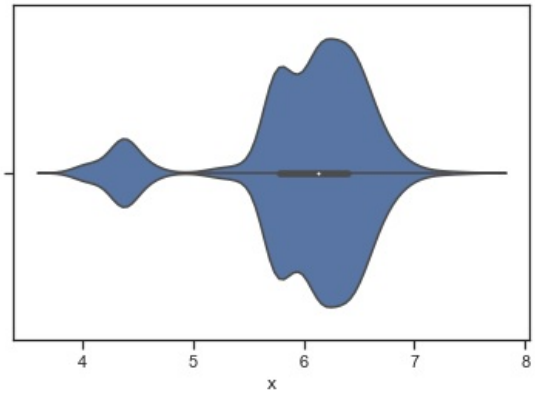

3250
3300
3350
3400
3450
3500
3550
3600
color_mean_price

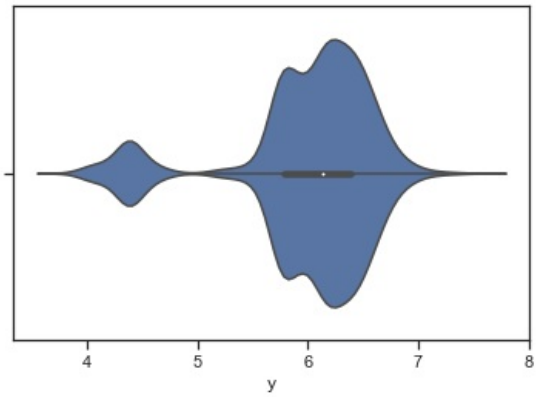

2800
3000
3200
3400
3600
3800
clarity_mean_price

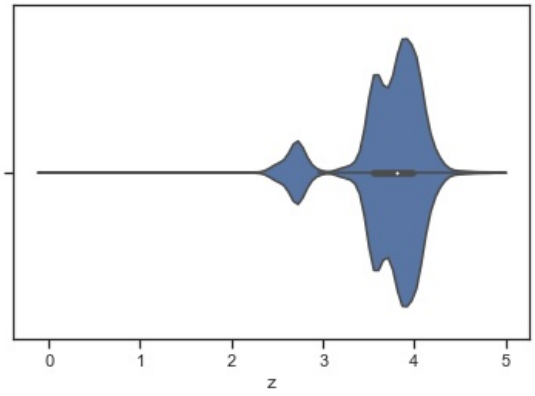

45
50
55
60
65
70
depth


50
55
60
65
70
table


0
1000
2000
3000
4000
5000
price


4
5
6
7
8
x


4
5
6
7
8
y


0
1
2
3
4
5
z

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

Для построения моделей будем использовать все признаки. Категориальные признаки закодированы.Выполним масштабирование данных.

In [17]:

```
# Числовые колонки для масштабирования
scale_cols = ['carat',
              'cut_mean_price',
              'color_mean_price',
              'clarity_mean_price',
              'depth',
              'table',
              'price',
              'x',
              'y',
              'z']
```

In [18]:

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

In [19]:

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

In [20]:

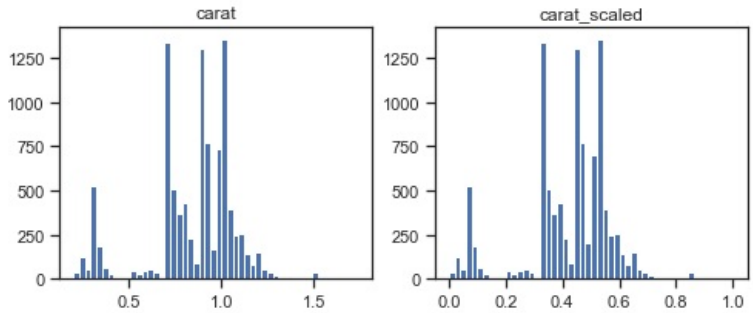
```
data.head()
```

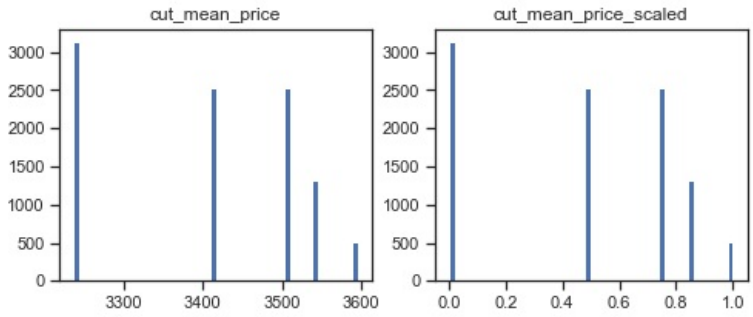
Out[20]:

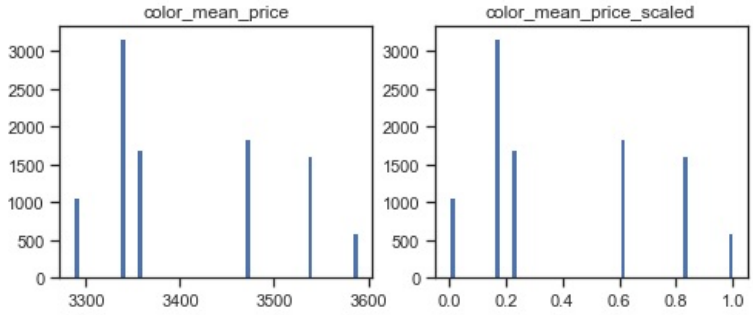
	carat	depth	table	price	x	y	z	cut_mean_price	color_mean_price	clarity_mean_price	carat_scaled	cut_mean_price_s
2	0.21	59.8	61.0	326	3.89	3.84	2.31	3503.800317	3338.471823	3430.389793	0.006494	0.7
3	0.23	56.9	65.0	327	4.05	4.07	2.31	3541.230008	3338.471823	3105.061500	0.019481	0.8
4	0.29	62.4	58.0	334	4.20	4.23	2.63	3503.800317	3286.895131	3141.847995	0.058442	0.7
5	0.31	63.3	58.0	335	4.34	4.35	2.75	3541.230008	3589.101161	3790.572568	0.071429	0.8
6	0.24	62.8	57.0	336	3.94	3.96	2.48	3414.363204	3589.101161	2912.161850	0.025974	0.4

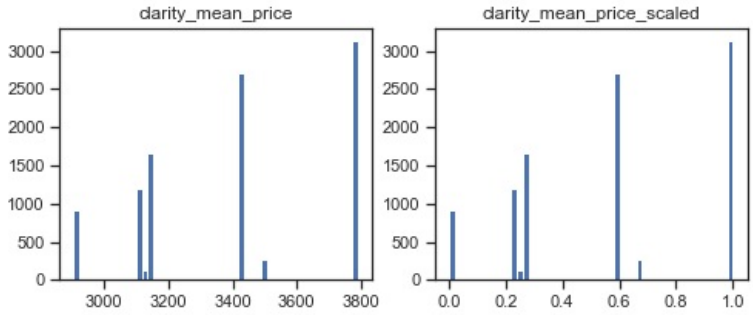
In [21]:

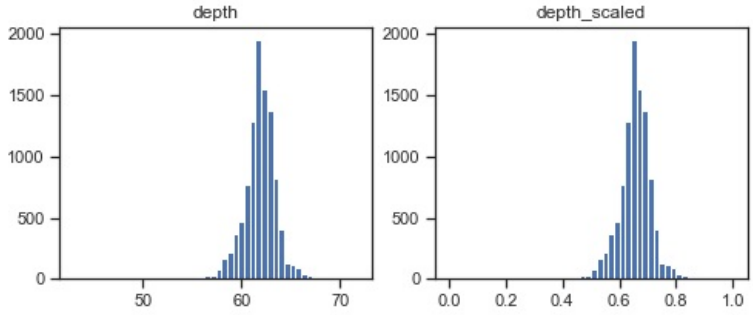
```
# Проверим, что масштабирование не повлияло на распределение данных
for col in scale_cols:
    col_scaled = col + '_scaled'

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




cut_mean_price
cut_mean_price_scaled


color_mean_price
color_mean_price_scaled


clarity_mean_price
clarity_mean_price_scaled


depth
depth_scaled

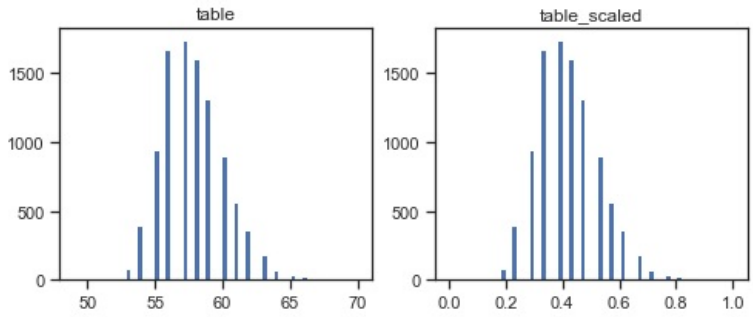
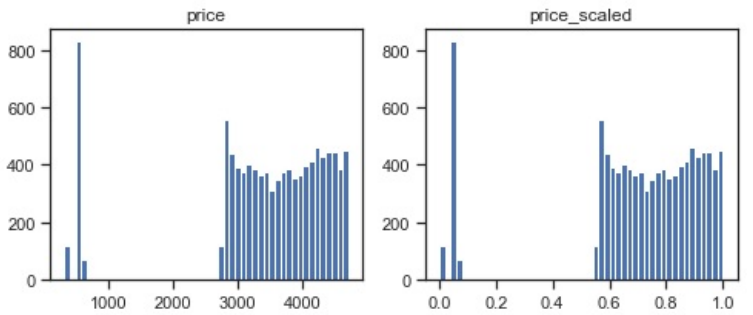
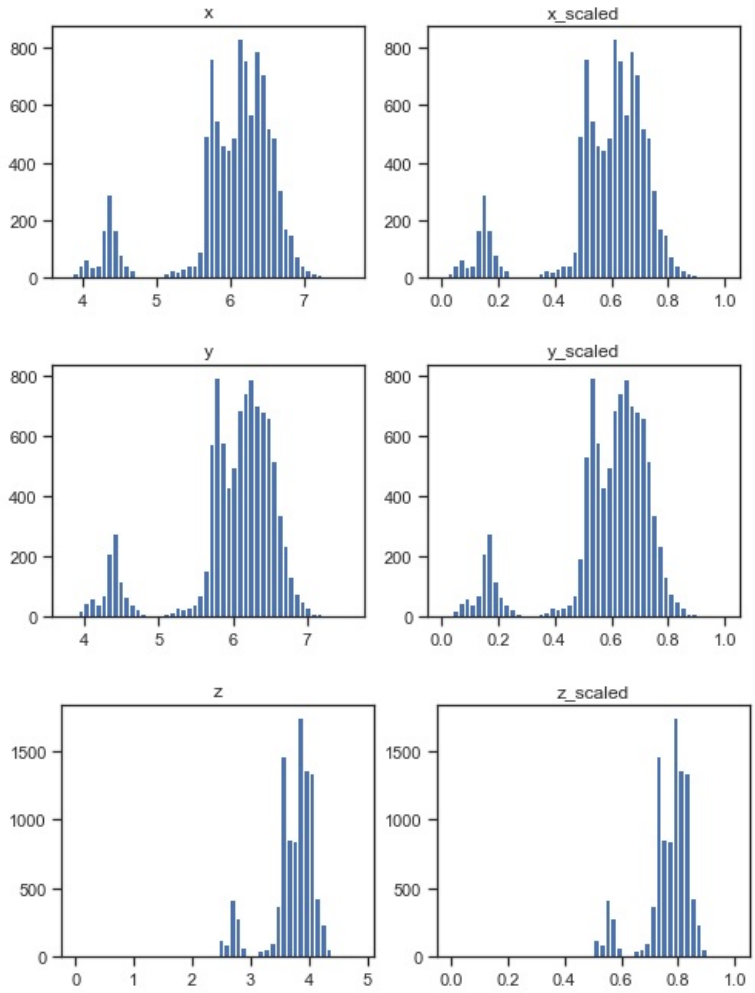


table
table_scaled


price
price_scaled



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

In [22]:

```
corr_cols_1 = scale_cols
corr_cols_1
```

Out[22]:

```
['carat',
 'cut_mean_price',
 'color_mean_price',
 'clarity_mean_price',
 'depth',
 'table',
 'price',
 'x',
 'y',
 'z']
```

In [23]:

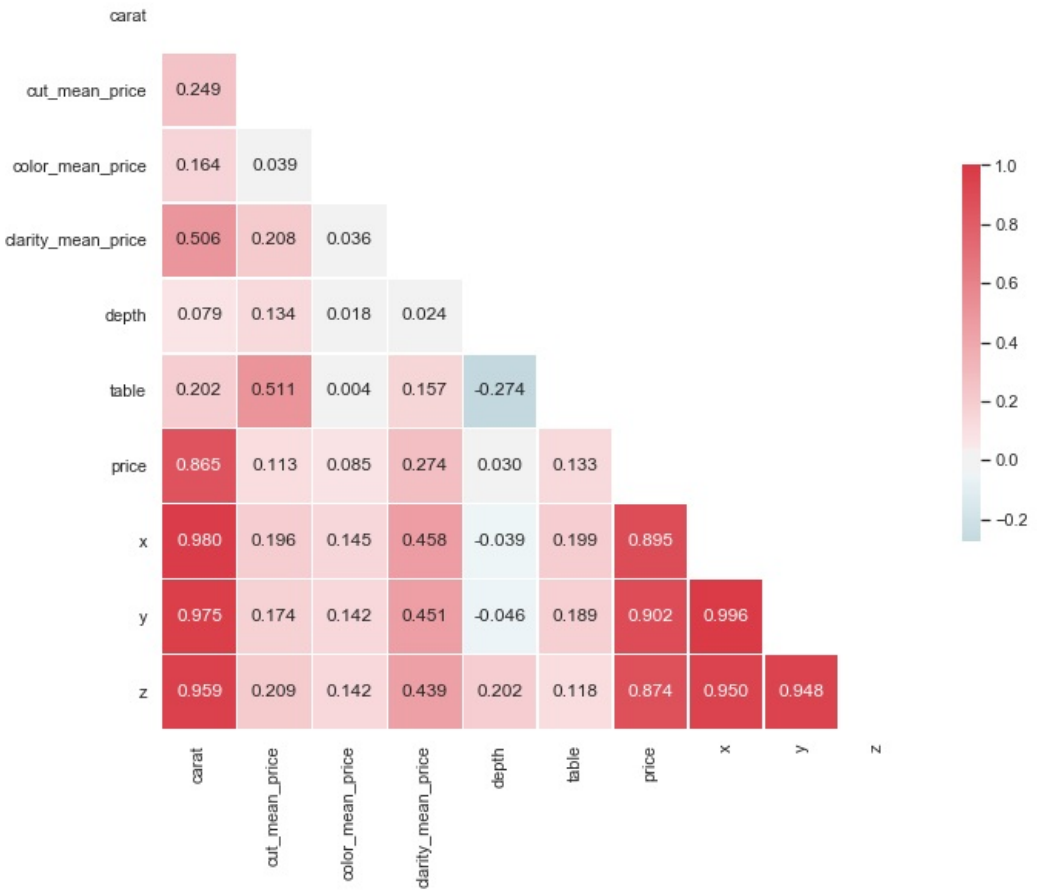
```
scale_cols_postfix = [x+'_scaled' for x in scale_cols]
corr_cols_2 = scale_cols_postfix
corr_cols_2
```

Out[23]:

```
['carat_scaled',
 'cut_mean_price_scaled',
 'color_mean_price_scaled',
 'clarity_mean_price_scaled',
 'depth_scaled',
 'table_scaled',
 'price_scaled',
 'x_scaled',
 'y_scaled',
 'z_scaled']
```

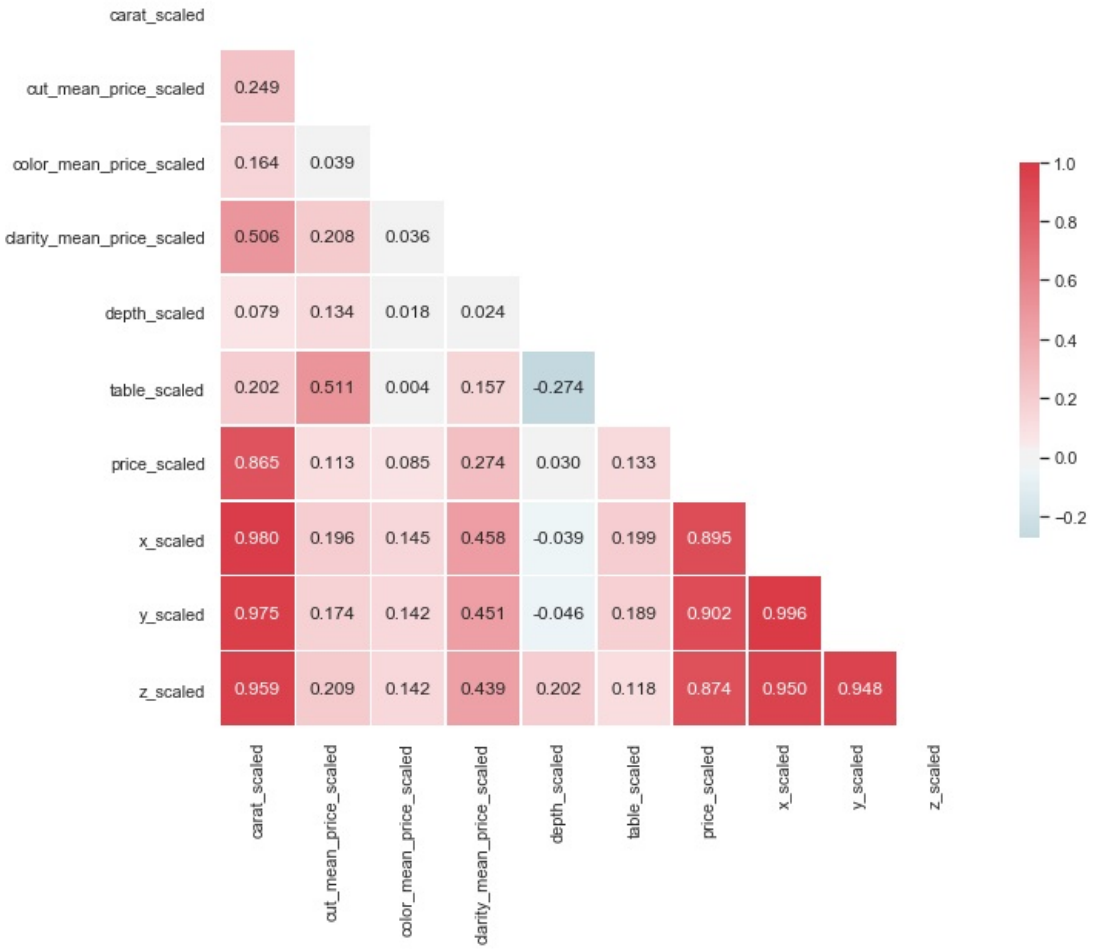
In [24]:

```
sns.set(style="white")
corr = data[corr_cols_1].corr()
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True
f, ax = plt.subplots(figsize=(11, 9))
cmap = sns.diverging_palette(220, 10, as_cmap=True)
g=sns.heatmap(corr, mask=mask, cmap=cmap, center=0, annot=True, fmt='.3f',
            square=True, linewidths=.5, cbar_kws={"shrink": .5})
```



In [25]:

```
sns.set(style="white")
corr = data[corr_cols_2].corr()
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True
f, ax = plt.subplots(figsize=(11, 9))
cmap = sns.diverging_palette(220, 10, as_cmap=True)
g=sns.heatmap(corr, mask=mask, cmap=cmap, center=0, annot=True, fmt='.3f',
            square=True, linewidths=.5, cbar_kws={"shrink": .5})
```



На основе корреляционной матрицы можно сделать следующие выводы:

- Корреляционные матрицы для исходных и масштабированных данных совпадают.
- Целевой признак регрессии "price" наиболее сильно коррелирует со следующими признаками:
  1. "carat" (0.865);
  2. "table" (0.133);
  3. "cut_mean_price" (0.113)
  4. "clarity_mean_price" (0.274) Эти признаки следует оставить в модели регрессии.
- Признаки "x" "y" "z" "carat" имеют большую корреляцию, поэтому все признаки не следует включать в модель. Будем использовать признак "carat".
- На основании корреляционной матрицы можно сделать вывод о том, что данные позволяют построить модель машинного обучения.

5) Выбор метрик для последующей оценки качества моделей

В качестве метрик для решения задачи регрессии будем использовать:

- Precision - доля верно предсказанных регрессором положительных объектов, из всех объектов, которые регрессор верно или неверно определил как положительные.
- Recall - доля верно предсказанных регрессором положительных объектов, из всех действительно положительных объектов.
- F_1-мера - для объединения precision и recall в единую метрику
- ROC AUC. Основана на вычислении следующих характеристик:
  - True Positive Rate, откладывается по оси ординат. Совпадает с recall.
  - False Positive Rate, откладывается по оси абсцисс. Показывает какую долю из объектов отрицательного класса алгоритм предсказал неверно.

In [26]:

```
class MetricLogger:

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

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

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

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

6) Выбор наиболее подходящих моделей для решения задачи реграссии

Для задачи регрессии будем использовать следующие модели:

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

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

In [27]:

```
task_regr_cols  = ['carat_scaled',
                   'clarity_mean_price_scaled',
                   'cut_mean_price_scaled',
                   'table_scaled'
                  ]
```

In [28]:

```
X = data[task_regr_cols]
Y = data['price_scaled']
X.shape
```

Out[28]:

```
(10000, 4)
```

In [29]:

```
# С использованием метода train_test_split разделим выборку на обучающую и тестовую
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25, random_state=1)
```

In [30]:

```
X_train.shape, X_test.shape, Y_train.shape, Y_test.shape
```

Out[30]:

```
((7500, 4), (2500, 4), (7500,), (2500,))
```

8) 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

In [31]:

```
# Модели
regr_models = {'LR': LinearRegression(),
               'KNN_5':KNeighborsRegressor(n_neighbors=5),
               'SVR':SVR(),
               'Tree':DecisionTreeRegressor(),
               'RF':RandomForestRegressor(),
               'GB':GradientBoostingRegressor()}
```

In [32]:

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

In [33]:

```
def regr_train_model(model_name, model, regrMetricLogger):
    model.fit(X_train, Y_train)
    Y_pred = model.predict(X_test)

    mae = mean_absolute_error(Y_test, Y_pred)
    mse = mean_squared_error(Y_test, Y_pred)
    r2 = r2_score(Y_test, Y_pred)

    regrMetricLogger.add('MAE', model_name, mae)
    regrMetricLogger.add('MSE', model_name, mse)
    regrMetricLogger.add('R2', model_name, r2)

    print('**************************************************')
    print(model)
    print()
    print('MAE={}, MSE={}, R2={}'.format(
        round(mae, 3), round(mse, 3), round(r2, 3)))
    print('**************************************************')
```

In [34]:

```
for model_name, model in regr_models.items():
    regr_train_model(model_name, model, regrMetricLogger)
```

```
**************************************************
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)

MAE=0.087, MSE=0.013, R2=0.799
**************************************************
**************************************************
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                    weights='uniform')

MAE=0.065, MSE=0.008, R2=0.884
**************************************************
**************************************************
SVR(C=1.0, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma='scale',
    kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False)

MAE=0.069, MSE=0.008, R2=0.883
**************************************************
**************************************************
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', 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')

MAE=0.067, MSE=0.009, R2=0.863
**************************************************
**************************************************
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      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)

MAE=0.063, MSE=0.007, R2=0.886
**************************************************
**************************************************
GradientBoostingRegressor(alpha=0.9, ccp_alpha=0.0, criterion='friedman_mse',
                          init=None, learning_rate=0.1, loss='ls', 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)

MAE=0.061, MSE=0.007, R2=0.9
**************************************************
```

9) Подбор гиперпараметров для выбранных моделей.

Метод ближайших соседей

In [35]:

```
n_range = np.array(range(1,1000,20))
tuned_parameters = [{'n_neighbors': n_range}]
tuned_parameters
```

Out[35]:

```
[{'n_neighbors': array([  1,  21,  41,  61,  81, 101, 121, 141, 161, 181, 201, 221, 241,
         261, 281, 301, 321, 341, 361, 381, 401, 421, 441, 461, 481, 501,
         521, 541, 561, 581, 601, 621, 641, 661, 681, 701, 721, 741, 761,
         781, 801, 821, 841, 861, 881, 901, 921, 941, 961, 981])}]
```

In [36]:

```
%%time
gs_KNN = GridSearchCV(KNeighborsRegressor(), tuned_parameters, cv=5, scoring='explained_variance')
gs_KNN.fit(X_train, Y_train)
```

```
CPU times: user 44.1 s, sys: 1.38 s, total: 45.5 s
Wall time: 46.7 s
```

Out[36]:

```
GridSearchCV(cv=5, error_score=nan,
             estimator=KNeighborsRegressor(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,  21,  41,  61,  81, 101, 121, 141, 161, 181, 20
1, 221, 241,
       261, 281, 301, 321, 341, 361, 381, 401, 421, 441, 461, 481, 501,
       521, 541, 561, 581, 601, 621, 641, 661, 681, 701, 721, 741, 761,
       781, 801, 821, 841, 861, 881, 901, 921, 941, 961, 981])}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='explained_variance', verbose=0)
```

In [37]:

```
# Лучшая модель
gs_KNN.best_estimator_
```

Out[37]:

```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=21, p=2,
                    weights='uniform')
```

In [38]:

```
# Лучшее значение параметров
gs_KNN.best_params_
```

Out[38]:

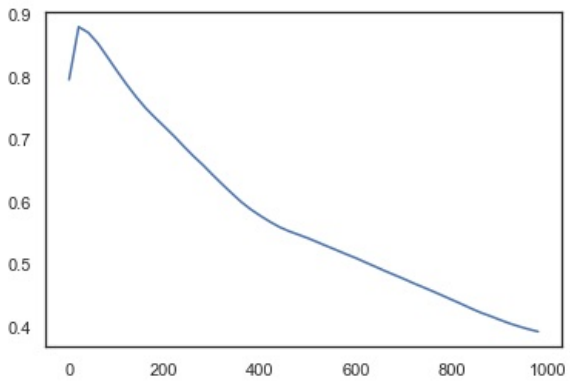
```
{'n_neighbors': 21}
```

In [39]:

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

Out[39]:

```
[<matplotlib.lines.Line2D at 0x1a32211590>]
```



Логистическая регрессия

In [40]:

```
grid = {'fit_intercept':[True,False], 'normalize':[True,False], 'copy_X':[True, False]}
gs_LR = GridSearchCV(LinearRegression(), grid, cv=2, scoring='explained_variance')
gs_LR.fit(X_train, Y_train)
```

Out[40]:

```
GridSearchCV(cv=2, error_score=nan,
             estimator=LinearRegression(copy_X=True, fit_intercept=True,
                                        n_jobs=None, normalize=False),
             iid='deprecated', n_jobs=None,
             param_grid={'copy_X': [True, False],
                         'fit_intercept': [True, False],
                         'normalize': [True, False]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='explained_variance', verbose=0)
```

In [41]:

```
# Лучшая модель
gs_LR.best_estimator_
```

Out[41]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=True)
```

In [42]:

```
# Лучшее значение параметров
gs_LR.best_params_
```

Out[42]:

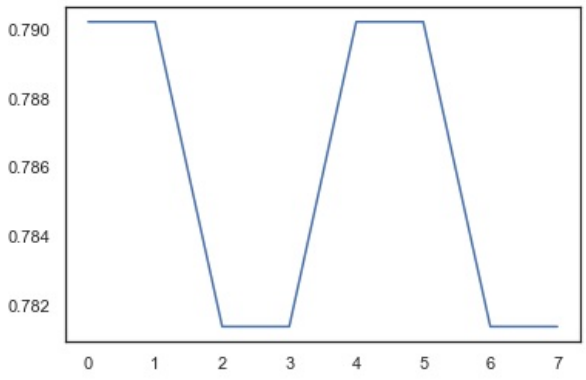
```
{'copy_X': True, 'fit_intercept': True, 'normalize': True}
```

In [43]:

```
# Изменение качества на тестовой выборке
plt.plot(gs_LR.cv_results_['mean_test_score'])
```

Out[43]:

```
[<matplotlib.lines.Line2D at 0x1a31ecde90>]
```



Машина опорных векторов

In [44]:

```
%%time
SVR_grid={'C':np.logspace(-3,4,12)}
gs_SVR = GridSearchCV(SVR(), SVR_grid, cv=2, scoring='explained_variance')
gs_SVR.fit(X_train, Y_train)
```

```
CPU times: user 4min 10s, sys: 2.22 s, total: 4min 13s
Wall time: 4min 16s
```

Out[44]:

```
GridSearchCV(cv=2, error_score=nan,
             estimator=SVR(C=1.0, cache_size=200, coef0=0.0, degree=3,
                           epsilon=0.1, gamma='scale', kernel='rbf',
                           max_iter=-1, shrinking=True, tol=0.001,
                           verbose=False),
             iid='deprecated', n_jobs=None,
             param_grid={'C': array([1.00000000e-03, 4.32876128e-03, 1.87381742e-02, 8.11130831e-02,
       3.51119173e-01, 1.51991108e+00, 6.57933225e+00, 2.84803587e+01,
       1.23284674e+02, 5.33669923e+02, 2.31012970e+03, 1.00000000e+04])},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='explained_variance', verbose=0)
```

In [45]:

```
# Лучшая модель
gs_SVR.best_estimator_
```

Out[45]:

```
SVR(C=28.48035868435799, cache_size=200, coef0=0.0, degree=3, epsilon=0.1,
    gamma='scale', kernel='rbf', max_iter=-1, shrinking=True, tol=0.001,
    verbose=False)
```

In [46]:

```
# Лучшее значение параметров
gs_SVR.best_params_
```

Out[46]:

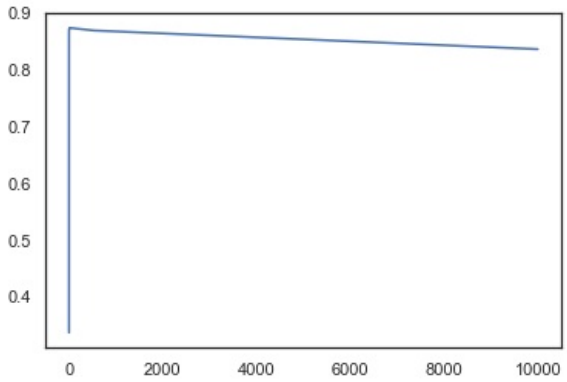
```
{'C': 28.48035868435799}
```

In [47]:

```
# Изменение качества на тестовой выборке
plt.plot(np.logspace(-3,4,12), gs_SVR.cv_results_['mean_test_score'])
```

Out[47]:

```
[<matplotlib.lines.Line2D at 0x1a32069ed0>]
```



Решающее дерево

In [48]:

```
%%time
tree_params={"max_depth":range(1,20), "max_features":range(1,4)}
gs_Tree = GridSearchCV(DecisionTreeRegressor(), tree_params, cv=5, scoring='explained_variance')
gs_Tree.fit(X_train, Y_train)
```

```
CPU times: user 1.82 s, sys: 34.7 ms, total: 1.85 s
Wall time: 1.93 s
```

Out[48]:

```
GridSearchCV(cv=5, error_score=nan,
             estimator=DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse',
                                             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'),
             iid='deprecated', n_jobs=None,
             param_grid={'max_depth': range(1, 20),
                         'max_features': range(1, 4)},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='explained_variance', verbose=0)
```

In [49]:

```
# Лучшая модель
gs_Tree.best_estimator_
```

Out[49]:

```
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=8,
                      max_features=3, 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')
```

In [50]:

```
# Лучшее значение параметров
gs_Tree.best_params_
```

Out[50]:

```
{'max_depth': 8, 'max_features': 3}
```

Случайный лес

In [51]:

```
%%time
RF_params={"max_leaf_nodes":range(2,12), "max_samples":range(2,22)}
gs_RF = GridSearchCV(RandomForestRegressor(), RF_params, cv=5, scoring='explained_variance')
gs_RF.fit(X_train, Y_train)
```

```
CPU times: user 1min 58s, sys: 2.1 s, total: 2min
Wall time: 2min 1s
```

Out[51]:

```
GridSearchCV(cv=5, error_score=nan,
             estimator=RandomForestRegressor(bootstrap=True, ccp_alpha=0.0,
                                             criterion='mse', 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),
             iid='deprecated', n_jobs=None,
             param_grid={'max_leaf_nodes': range(2, 12),
                         'max_samples': range(2, 22)},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='explained_variance', verbose=0)
```

In [52]:

```
# Лучшая модель
gs_RF.best_estimator_
```

Out[52]:

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nodes=10,
                      max_samples=21, 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)
```

In [53]:

```
# Лучшее значение параметров
gs_RF.best_params_
```

Out[53]:

```
{'max_leaf_nodes': 10, 'max_samples': 21}
```

Градиентный бустинг

In [54]:

```
%%time
GB_params={"max_features":range(1,4), "max_leaf_nodes":range(2,22)}
gs_GB = GridSearchCV(GradientBoostingRegressor(), GB_params, cv=5, scoring='explained_variance')
gs_GB.fit(X_train, Y_train)
```

```
CPU times: user 40.6 s, sys: 569 ms, total: 41.2 s
Wall time: 41.8 s
```

Out[54]:

```
GridSearchCV(cv=5, error_score=nan,
             estimator=GradientBoostingRegressor(alpha=0.9, ccp_alpha=0.0,
                                                 criterion='friedman_mse',
                                                 init=None, learning_rate=0.1,
                                                 loss='ls', 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),
             iid='deprecated', n_jobs=None,
             param_grid={'max_features': range(1, 4),
                         'max_leaf_nodes': range(2, 22)},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='explained_variance', verbose=0)
```

In [55]:

```
# Лучшая модель
gs_GB.best_estimator_
```

Out[55]:

```
GradientBoostingRegressor(alpha=0.9, ccp_alpha=0.0, criterion='friedman_mse',
                          init=None, learning_rate=0.1, loss='ls', max_depth=3,
                          max_features=3, max_leaf_nodes=16,
                          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)
```

In [56]:

```
# Лучшее значение параметров
gs_GB.best_params_
```

Out[56]:

```
{'max_features': 3, 'max_leaf_nodes': 16}
```

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

In [57]:

```
models_grid = { 'LR_new':gs_LR.best_estimator_,
                'KNN_new':gs_KNN.best_estimator_,
                 'SVR_new':gs_SVR.best_estimator_,
               'Tree_new':gs_Tree.best_estimator_,
                'RF_new':gs_RF.best_estimator_,
               'GB_new':gs_GB.best_estimator_
               }
```

In [58]:

```
for model_name, model in models_grid.items():
    regr_train_model(model_name, model, regrMetricLogger)
```

```
**************************************************
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=True)

MAE=0.087, MSE=0.013, R2=0.799
**************************************************
**************************************************
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=21, p=2,
                    weights='uniform')

MAE=0.064, MSE=0.007, R2=0.89
**************************************************
**************************************************
SVR(C=28.48035868435799, cache_size=200, coef0=0.0, degree=3, epsilon=0.1,
    gamma='scale', kernel='rbf', max_iter=-1, shrinking=True, tol=0.001,
    verbose=False)

MAE=0.068, MSE=0.007, R2=0.885
**************************************************
**************************************************
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=8,
                      max_features=3, 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')

MAE=0.063, MSE=0.007, R2=0.889
**************************************************
**************************************************
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nodes=10,
                      max_samples=21, 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)

MAE=0.073, MSE=0.008, R2=0.874
**************************************************
**************************************************
GradientBoostingRegressor(alpha=0.9, ccp_alpha=0.0, criterion='friedman_mse',
                          init=None, learning_rate=0.1, loss='ls', max_depth=3,
                          max_features=3, max_leaf_nodes=16,
                          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)

MAE=0.061, MSE=0.007, R2=0.9
**************************************************
```

11) Формирование выводов о качестве построенных моделей на основе выбранных метрик.

In [59]:

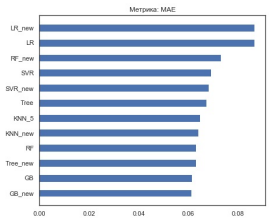
```
# Метрики качества модели
regr_metrics = regrMetricLogger.df['metric'].unique()
regr_metrics
```

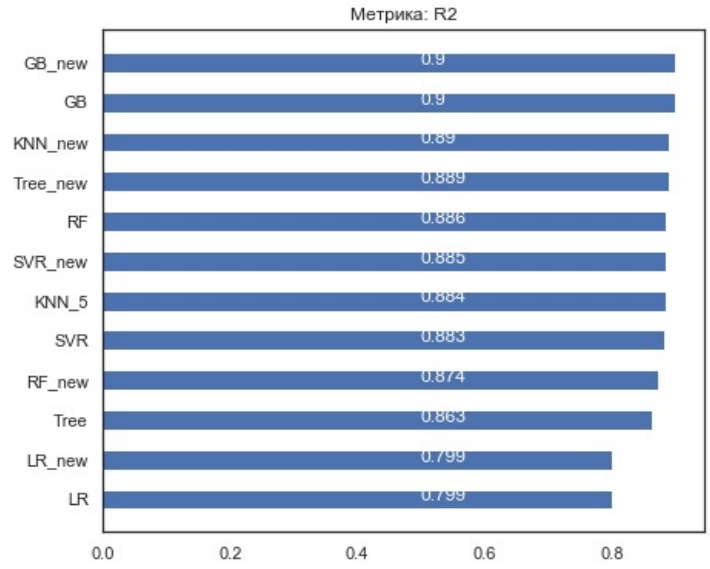
Out[59]:

```
array(['MAE', 'MSE', 'R2'], dtype=object)
```

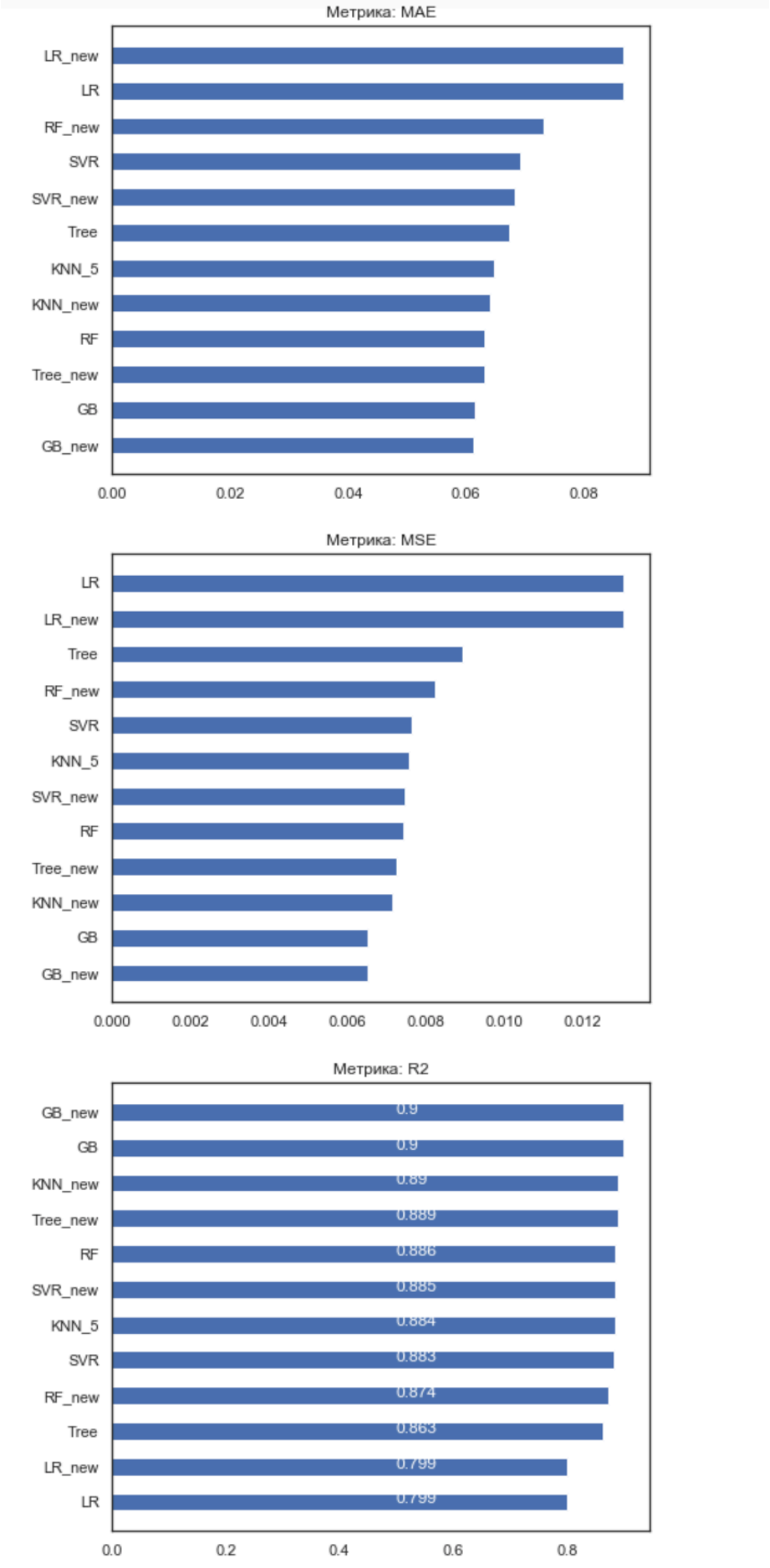
In [61]:

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





Вывод: на основании всех четырех метрик, лучшей оказалась модель "Градиентный бустинг".


Метрика: MAE
LR_new
LR
RF_new
SVR
SVR_new
Tree
KNN_5
KNN_new
RF
Tree_new
GB
GB_new
0.00
0.02
0.04
0.06
0.08
Метрика: MSE
LR
LR_new
Tree
RF_new
SVR
KNN_5
SVR_new
RF
Tree_new
KNN_new
GB
GB_new
0.000
0.002
0.004
0.006
0.008
0.010
0.012
Метрика: R2
GB_new
GB
KNN_new
Tree_new
RF
SVR_new
KNN_5
SVR
RF_new
Tree
LR_new
LR
0.9
0.9
0.89
0.889
0.886
0.885
0.884
0.883
0.874
0.863
0.799
0.799
0.0
0.2
0.4
0.6
0.8

Заключение

В данном курсовом проекте была решена типовая задача машинного обучения. Был выбран набор данных для построения моделей машинного обучения, проведен разведочный анализ данных и построены графики, необходимые для понимания структуры данных. Были выбраны признаки, подходящие для построения моделей, масштабированы данные и проведен корреляционный анализ данных. Это позволило сформировать промежуточные выводы о возможности построения моделей машинного обучения.

На следующем этапе были выбраны метрики для последующей оценки качества моделей и наиболее подходящие модели для решения задачи регрессии. Затем были сформированы обучающая и тестовая выборки на основе исходного набора данных и построено базовое решение для выбранных моделей без подбора гиперпараметров.

Следующим шагом был подбор гиперпараметров для выбранных моделей, после чего мы смогли сравнить качество полученных моделей с качеством baseline-моделей. Большинство моделей, для которых были подобраны оптимальные значения гиперпараметров, показали лучший результат.

В заключение, были сформированы выводы о качестве построенных моделей на основе выбранных метрик. Для наглядности результаты сравнения качества были отображены в виде графиков, а также сделаны выводы в форме текстового описания. Четыре метрики показали, что для выбранного набора данных лучшей моделью оказалась «градиентный бустинг».

Список использованных источников

1. Ю.Е. Гапанюк, Лекции по курсу «Технологии машинного обучения» 2019-2020 учебный год.
2. scikit-learn Machine Learning in Python: [сайт]. URL: https://scikit-learn.org/stable/
3. Diamonds [Электронный ресурс]. URL: https://www.kaggle.com/shivam2503/diamonds