```
import numpy as np
import pandas as pd
from typing import Dict, Tuple
from scipy import stats
from IPython.display import Image
from sklearn.externals.six import StringIO
from IPython.display import Image
import graphviz
import pydotplus
from sklearn.datasets import load_iris, load_boston
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor, KNeighborsClassifier
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
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.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.ensemble import BaggingClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import mean_absolute_error, mean_squared_error, mean_squared_log_erro
from sklearn.metrics import roc_curve, roc_auc_score
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
sns.set(style="ticks")
from sklearn.svm import SVC, NuSVC, LinearSVC, OneClassSVM, SVR, NuSVR, LinearSVR
%matplotlib inline
sns.set(rc={'figure.figsize':(16,8)})
```

```
for col in df.columns:
    count = df[df[col].isnull()].shape[0]
    print('{}  -  {}'.format(col, count))
print ('{} - размер датасета'.format(df.shape))
```

```
longitude  -  0
latitude  -  0
housing_median_age  -  0
total_rooms  -  0
total_bedrooms  -  0
population  -  0
households  -  0
median_income  -  0
median_house_value  -  0
ocean_proximity  -  0
(20433, 10) - размер датасета
```

```
df = pd.read_csv('housing.csv', error_bad_lines=False, comment='#')
df = df.dropna(axis=0, how='any')
```

```
for col in df.columns:
    count = df[df[col].isnull()].shape[0]
    print('{}  -  {}'.format(col, count))
print ('{} - размер датасета'.format(df.shape))
#df.drop(columns='id', inplace = True);
#df.drop(columns='Cabin', inplace= True)
```

```
longitude  -  0
latitude  -  0
housing_median_age  -  0
total_rooms  -  0
total_bedrooms  -  0
population  -  0
households  -  0
median_income  -  0
median_house_value  -  0
ocean_proximity  -  0
(20433, 10) - размер датасета
```

```
df
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	populati
0	-122.23	37.88	41.0	880.0	129.0	32
1	-122.22	37.86	21.0	7099.0	1106.0	240
2	-122.24	37.85	52.0	1467.0	190.0	49
3	-122.25	37.85	52.0	1274.0	235.0	55
4	-122.25	37.85	52.0	1627.0	280.0	56
...	...	...	...	...	...	
20635	-121.09	39.48	25.0	1665.0	374.0	84
20636	-121.21	39.49	18.0	697.0	150.0	35
20637	-121.22	39.43	17.0	2254.0	485.0	100
20638	-121.32	39.43	18.0	1860.0	409.0	74
20639	-121.24	39.37	16.0	2785.0	616.0	138

20433 rows × 10 columns

```
df.reset_index(drop=True, inplace=True)
```

Кодируем категориальные признаки

```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
le = LabelEncoder()
df['ocean_proximity'].unique()
```

```
array(['NEAR BAY', '<1H OCEAN', 'INLAND', 'NEAR OCEAN', 'ISLAND'],
      dtype=object)
```

```
oc = le.fit_transform(df['ocean_proximity'])
ocean = pd.DataFrame({'ocean_proximity' : oc.T})
np.unique(oc)
```

```
array([0, 1, 2, 3, 4])
```

```
del df['ocean_proximity']
df = df.join(ocean)
df
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	populati
0	-122.23	37.88	41.0	880.0	129.0	32
1	-122.22	37.86	21.0	7099.0	1106.0	240
2	-122.24	37.85	52.0	1467.0	190.0	49
3	-122.25	37.85	52.0	1274.0	235.0	55
4	-122.25	37.85	52.0	1627.0	280.0	56
...	...	...	...	...	...	
20428	-121.09	39.48	25.0	1665.0	374.0	84
20429	-121.21	39.49	18.0	697.0	150.0	35
20430	-121.22	39.43	17.0	2254.0	485.0	100
20431	-121.32	39.43	18.0	1860.0	409.0	74
20432	-121.24	39.37	16.0	2785.0	616.0	138

20433 rows × 10 columns

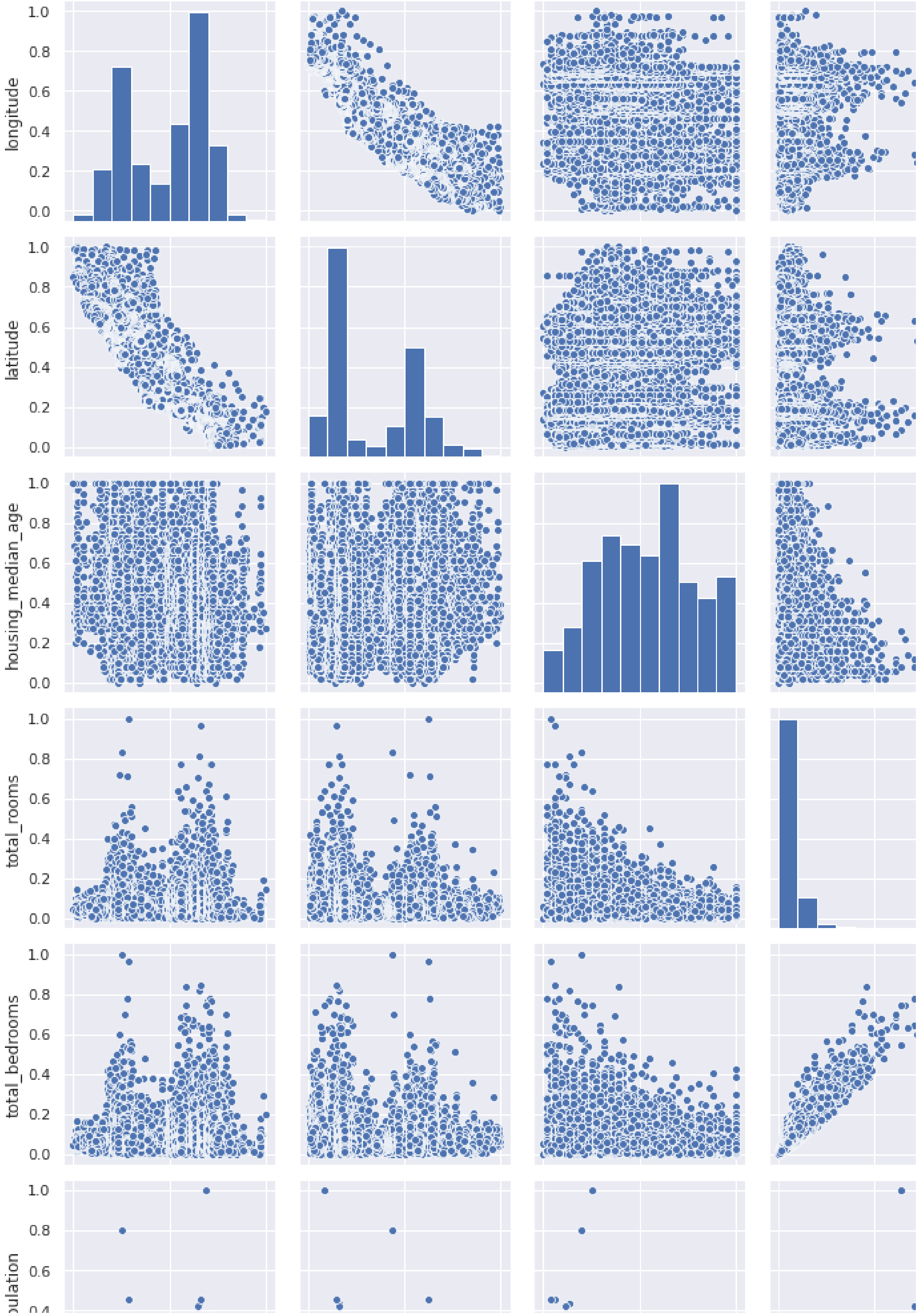
```
from sklearn.preprocessing import MinMaxScaler
features
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	populati
0	-122.23	37.88	41.0	880.0	129.0	322
1	-122.22	37.86	21.0	7099.0	1106.0	240
2	-122.24	37.85	52.0	1467.0	190.0	49
4	-122.25	37.85	52.0	1627.0	280.0	56

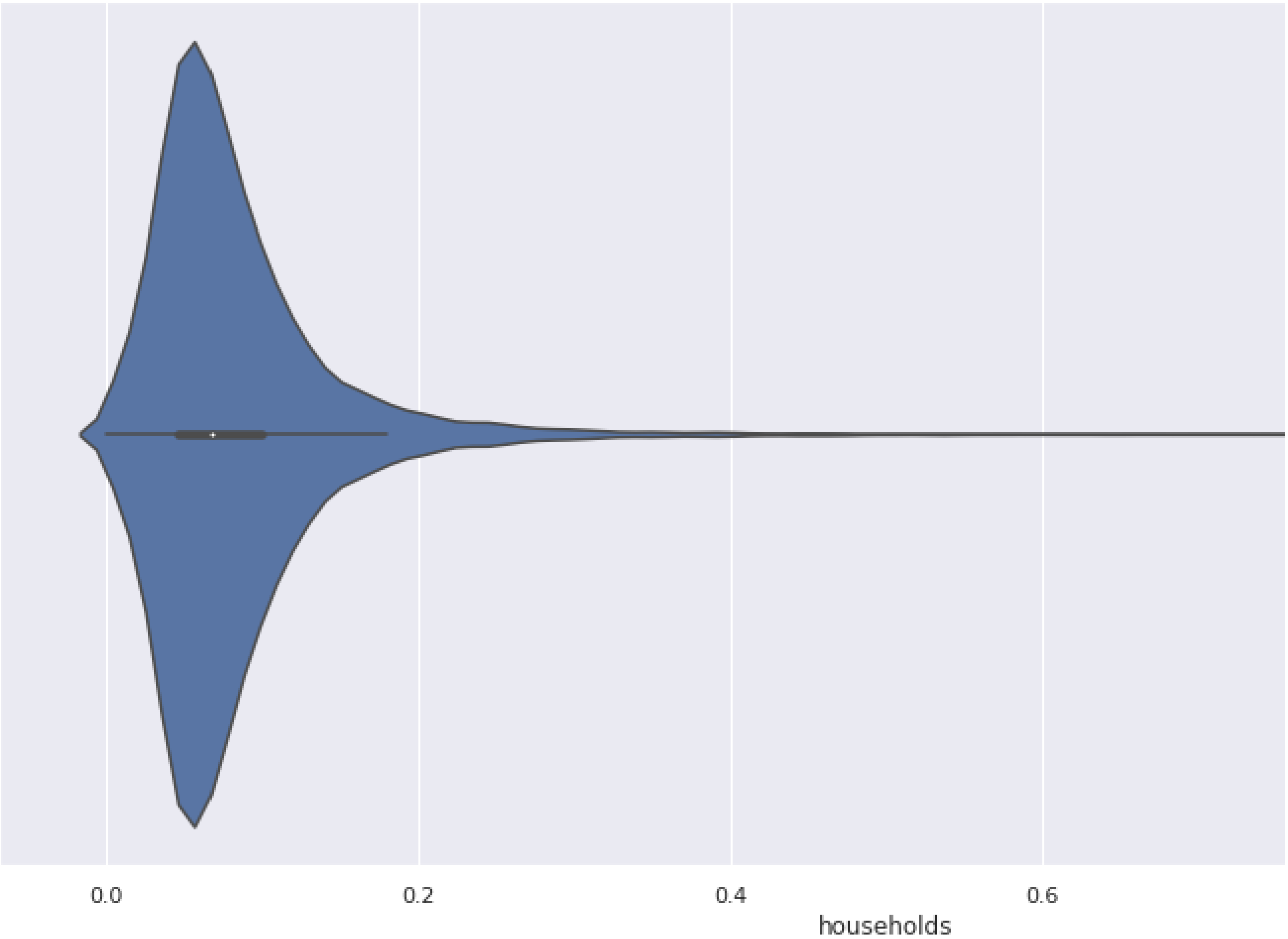
Построим парные статистики

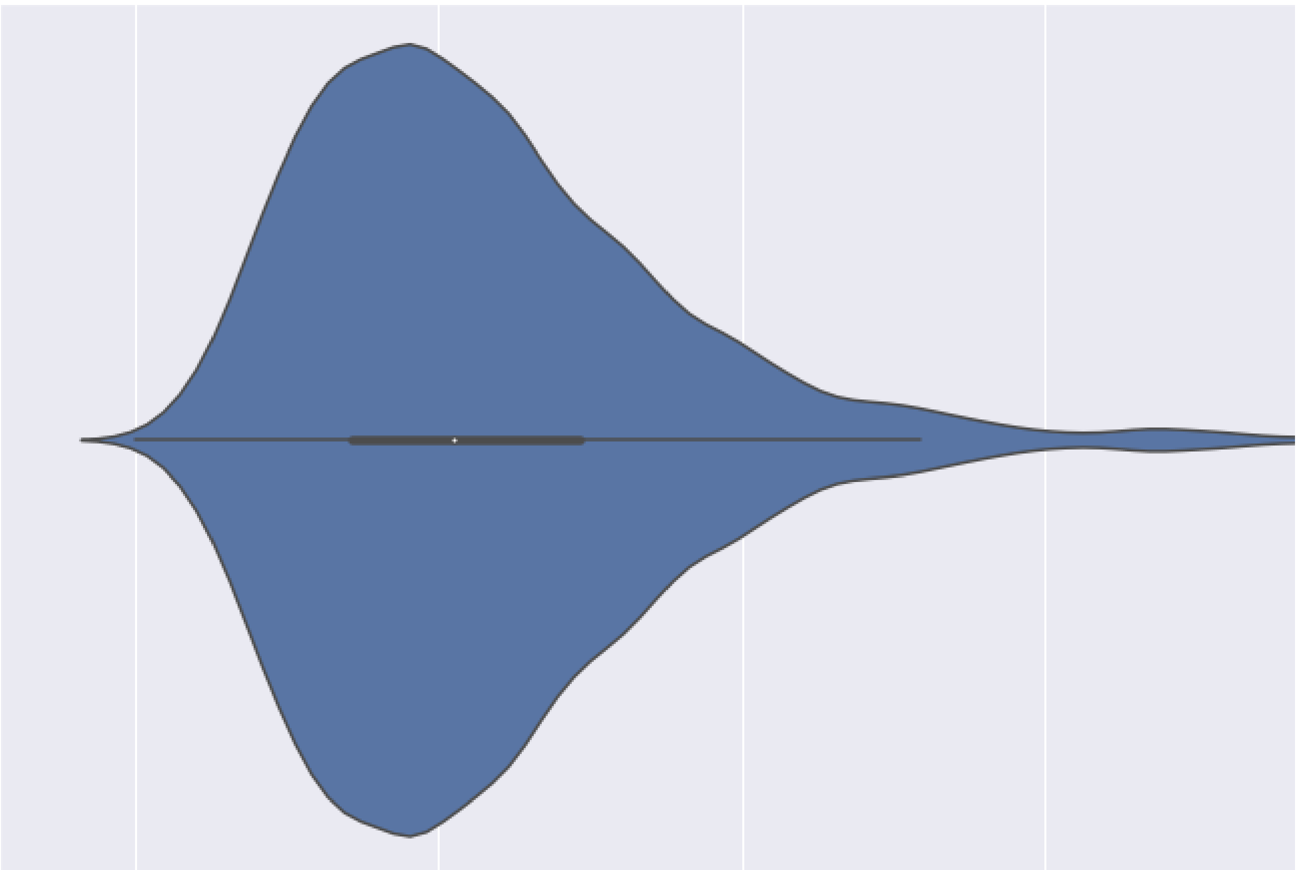
```
sns.pairplot(df)
```

```
<seaborn.axisgrid.PairGrid at 0x7ff4174d4f60>
```




```
# Скрипичные диаграммы для числовых колонок
for col in ['households', 'median_income', 'median_house_value','ocean_proximity']:
    sns.violinplot(x=df[col])
    plt.show()
```





Произведем масштабирование

```
scal= MinMaxScaler()
s = scal.fit_transform(df)
```

```
df = pd.DataFrame(s, columns= ['longitude', 'latitude', 'housing_median_age', 'total_rooms
df
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	populati
0	0.211155	0.567481	0.784314	0.022331	0.019863	0.0089
1	0.212151	0.565356	0.392157	0.180503	0.171477	0.0672
2	0.210159	0.564293	1.000000	0.037260	0.029330	0.0138
3	0.209163	0.564293	1.000000	0.032352	0.036313	0.0155
4	0.209163	0.564293	1.000000	0.041330	0.043296	0.0157
...	...	...	...	...	...	
20428	0.324701	0.737513	0.470588	0.042296	0.057883	0.0235
20429	0.312749	0.738576	0.333333	0.017676	0.023122	0.0098
20430	0.311753	0.732200	0.313725	0.057277	0.075109	0.0281
20431	0.301793	0.732200	0.333333	0.047256	0.063315	0.0206
20432	0.309761	0.725824	0.294118	0.070782	0.095438	0.0387

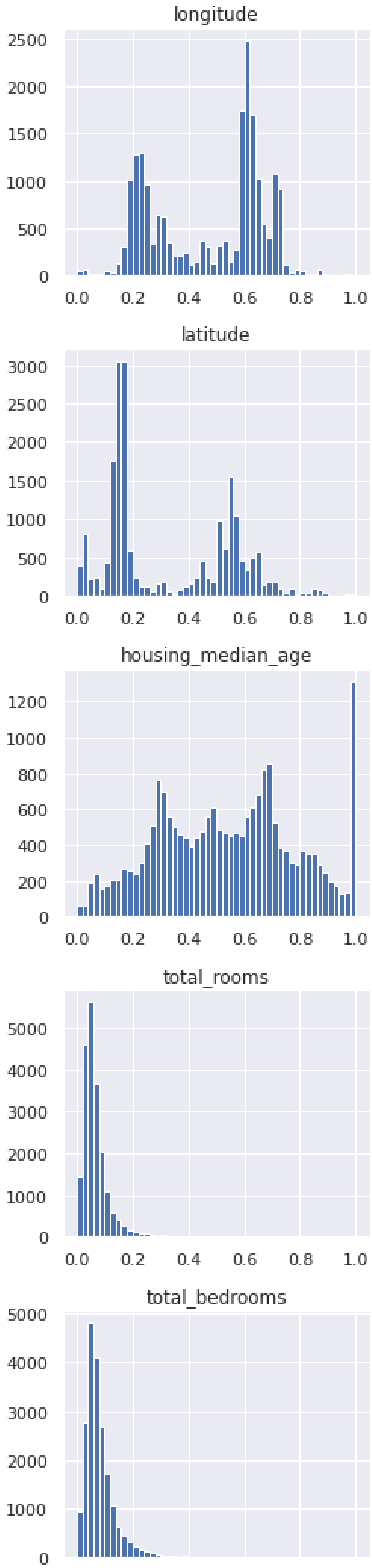
20433 rows × 10 columns

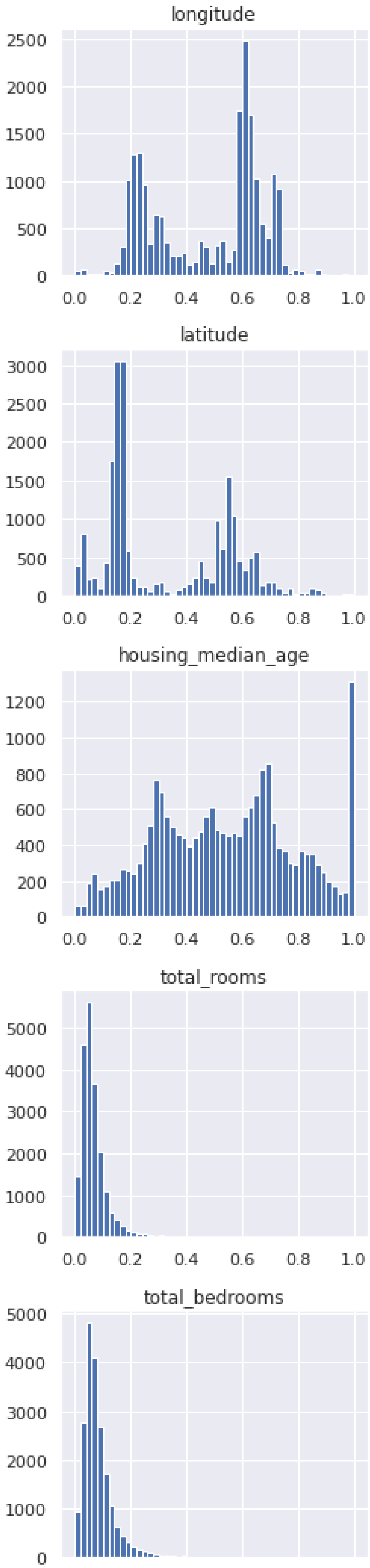
```
features = df.drop(columns='median_house_value')
target = df.median_house_value
```

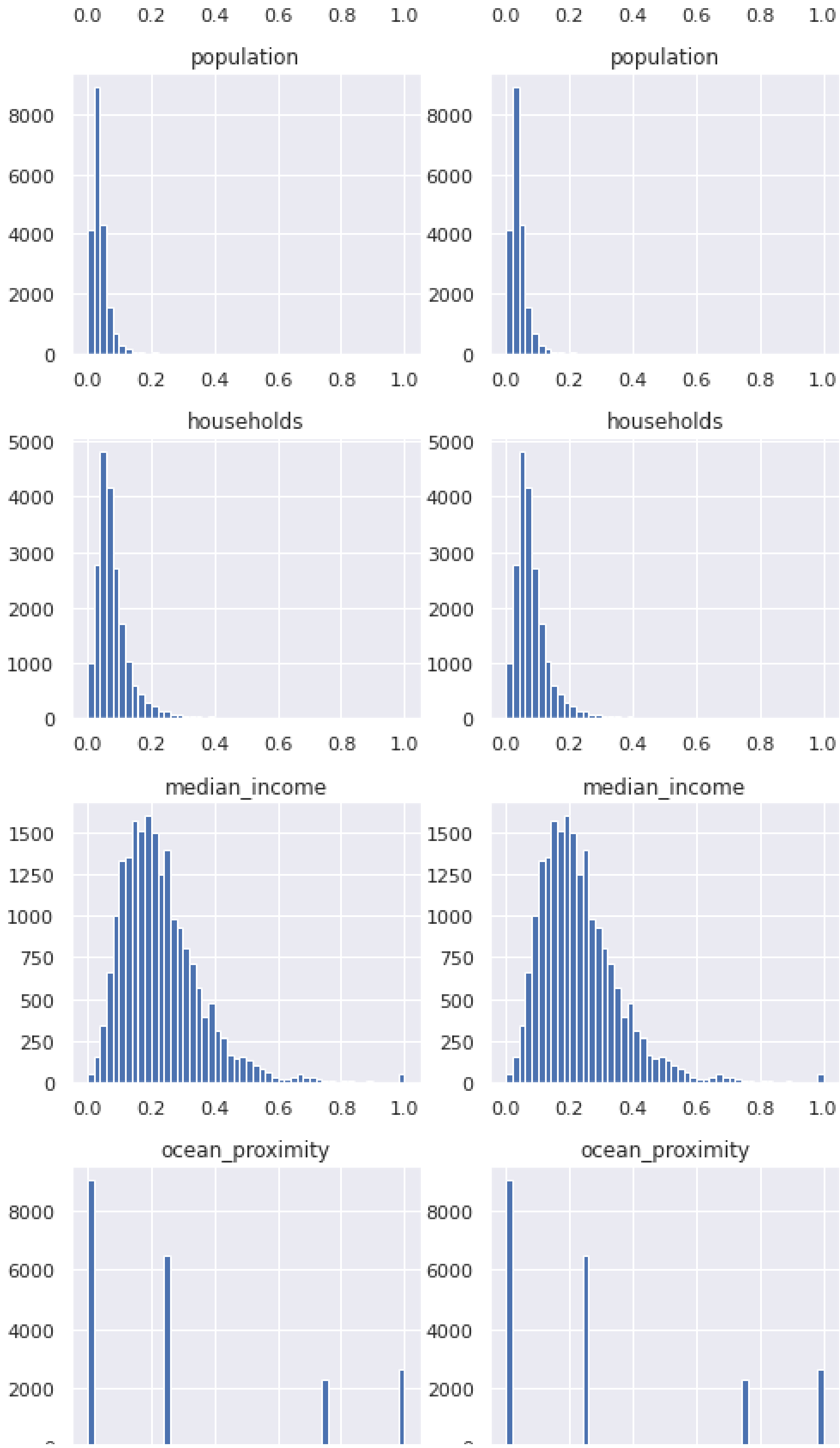
```
0        0.902266
1        0.708247
2        0.695051
3        0.672783
4        0.674638
           ...
20428    0.130105
20429    0.128043
20430    0.159383
20431    0.143713
20432    0.153403
Name: median_house_value, Length: 20433, dtype: float64
```

```
# Проверим, что масштабирование не повлияло на распределение данных
for col in features:
    col_scaled = col

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


longitude
latitude
housing_median_age
total_rooms
total_bedrooms


longitude
latitude
housing_median_age
total_rooms
total_bedrooms



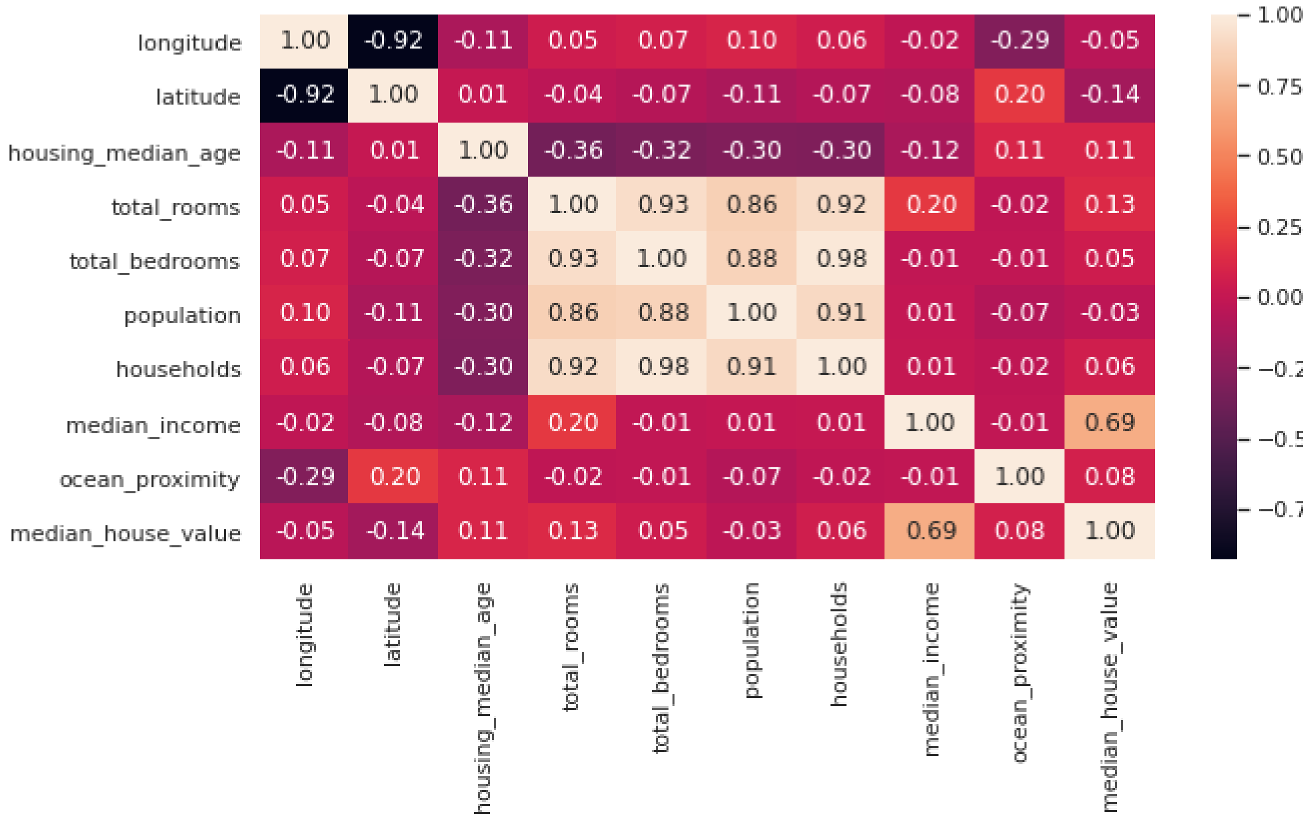

```
X_train, X_test, y_train, y_test = train_test_split(
    features, target, test_size=0.2, random_state = 1)
```

```
target_df = pd.DataFrame(target, columns=['median_house_value'])
df_merge = features.join(target_df)
df_merge
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	populati
0	0.211155	0.567481	0.784314	0.022331	0.019863	0.0089
1	0.212151	0.565356	0.392157	0.180503	0.171477	0.0672
2	0.210159	0.564293	1.000000	0.037260	0.029330	0.0138
3	0.209163	0.564293	1.000000	0.032352	0.036313	0.0155
4	0.209163	0.564293	1.000000	0.041330	0.043296	0.0157
...	...	...	...	...	...	
20428	0.324701	0.737513	0.470588	0.042296	0.057883	0.0235
20429	0.312749	0.738576	0.333333	0.017676	0.023122	0.0098
20430	0.311753	0.732200	0.313725	0.057277	0.075109	0.0281
20431	0.301793	0.732200	0.333333	0.047256	0.063315	0.0206
20432	0.309761	0.725824	0.294118	0.070782	0.095438	0.0387

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

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff409489be0>
```




```
# Выборки для задачи регресии
regr_X_train = X_train
regr_X_test = X_test
regr_Y_train = y_train
regr_Y_test = y_test
```

```
regr_X_train.shape, regr_X_test.shape, regr_Y_train.shape, regr_Y_test.shape
```

```
((16346, 9), (4087, 9), (16346,), (4087,))
```

```
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, inp
        # 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()
```

```
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, export_graphviz
from sklearn.svm import SVC, NuSVC, LinearSVC, OneClassSVM, SVR, NuSVR, LinearSVR
from sklearn.ensemble import ExtraTreesClassifier, ExtraTreesRegressor
from sklearn.ensemble import GradientBoostingClassifier, GradientBoostingRegressor

# Модели
regr_models = {'LR': LinearRegression()
```

```
regr_models = {'LR': LinearRegression(),
               'KNN_5':KNeighborsRegressor(n_neighbors=5),
               'SVR':SVR(),
               'Tree':DecisionTreeRegressor(),
               'RF':RandomForestRegressor(),
               'GB':GradientBoostingRegressor()}
# Модели
clas_models = {'LogR': LogisticRegression(),
               'KNN_5':KNeighborsClassifier(n_neighbors=5),
               'SVC':SVC(),
               'Tree':DecisionTreeClassifier(),
               'RF':RandomForestClassifier(),
               'GB':GradientBoostingClassifier()}


# Сохранение метрик
regrMetricLogger = MetricLogger()


def regr_train_model(model_name, model, regrMetricLogger):
    model.fit(regr_X_train, regr_Y_train)
    Y_pred = model.predict(regr_X_test)

    mae = mean_absolute_error(regr_Y_test, Y_pred)
    mse = mean_squared_error(regr_Y_test, Y_pred)
    r2 = r2_score(regr_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('*****************************************************')
```

Построим несколько моделей МО

1. Линейная регрессия
2. Метод ближайших соседей
3. Метод опорных векторов
4. Случайный лес
5. Решающее дерево
6. Градиентный бустинг

```
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=0.104, MSE=0.02, R2=0.645
****************************************************
****************************************************
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                    weights='uniform')

MAE=0.085, MSE=0.016, R2=0.714
****************************************************
****************************************************
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=0.085, MSE=0.015, R2=0.741
****************************************************
****************************************************
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=0.087, MSE=0.019, R2=0.671
****************************************************
****************************************************
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=0.01, R2=0.827
****************************************************
****************************************************
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)
```

Подбор гиперпараметров

KNeighbors

```
n_range = np.array(range(1,100,5))
tuned_parameters = [{'n_neighbors': n_range}]
tuned_parameters
```

```
[{'n_neighbors': array([ 1,  6, 11, 16, 21, 26, 31, 36, 41, 46, 51, 56, 61, 66, 71, 7
         86, 91, 96])}]
```

```
regr_gs_KN = GridSearchCV(KNeighborsRegressor(), tuned_parameters, cv=5, scoring='neg_mean
regr_gs_KN.fit(regr_X_train, regr_Y_train)
```

```
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,  6, 11, 16, 21, 26, 31, 36, 41, 4
       86, 91, 96])}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='neg_mean_squared_error', verbose=0)
```

```
# Лучшая модель
regr_gs_KN.best_estimator_
```

```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=11, p=2,
                    weights='uniform')
```

```
# Лучшее значение параметров
regr_gs_KN.best_params_
```

```
{'n_neighbors': 11}
```

```
# Изменение качества на тестовой выборке в зависимости от K-соседей
plt.plot(n_range, regr_gs_KN.cv_results_['mean_test_score'])
```

```
[<matplotlib.lines.Line2D at 0x7ff406bf5630>]
```

DecisionTree

```
n_range = np.array(range(1,50,3))
tuned_parameters = [{'max_depth': n_range}]
tuned_parameters
```

```
[{'max_depth': array([ 1,  4,  7, 10, 13, 16, 19, 22, 25, 28, 31, 34, 37, 40, 43, 46,
```

```
regr_gs_DT = GridSearchCV(DecisionTreeRegressor(), tuned_parameters, cv=5, scoring='neg_me
regr_gs_DT.fit(regr_X_train, regr_Y_train)
```

```
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': array([ 1,  4,  7, 10, 13, 16, 19, 22, 25, 28,
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='neg_mean_squared_error', verbose=0)
```

```
# Лучшая модель
regr_gs_DT.best_estimator_
```

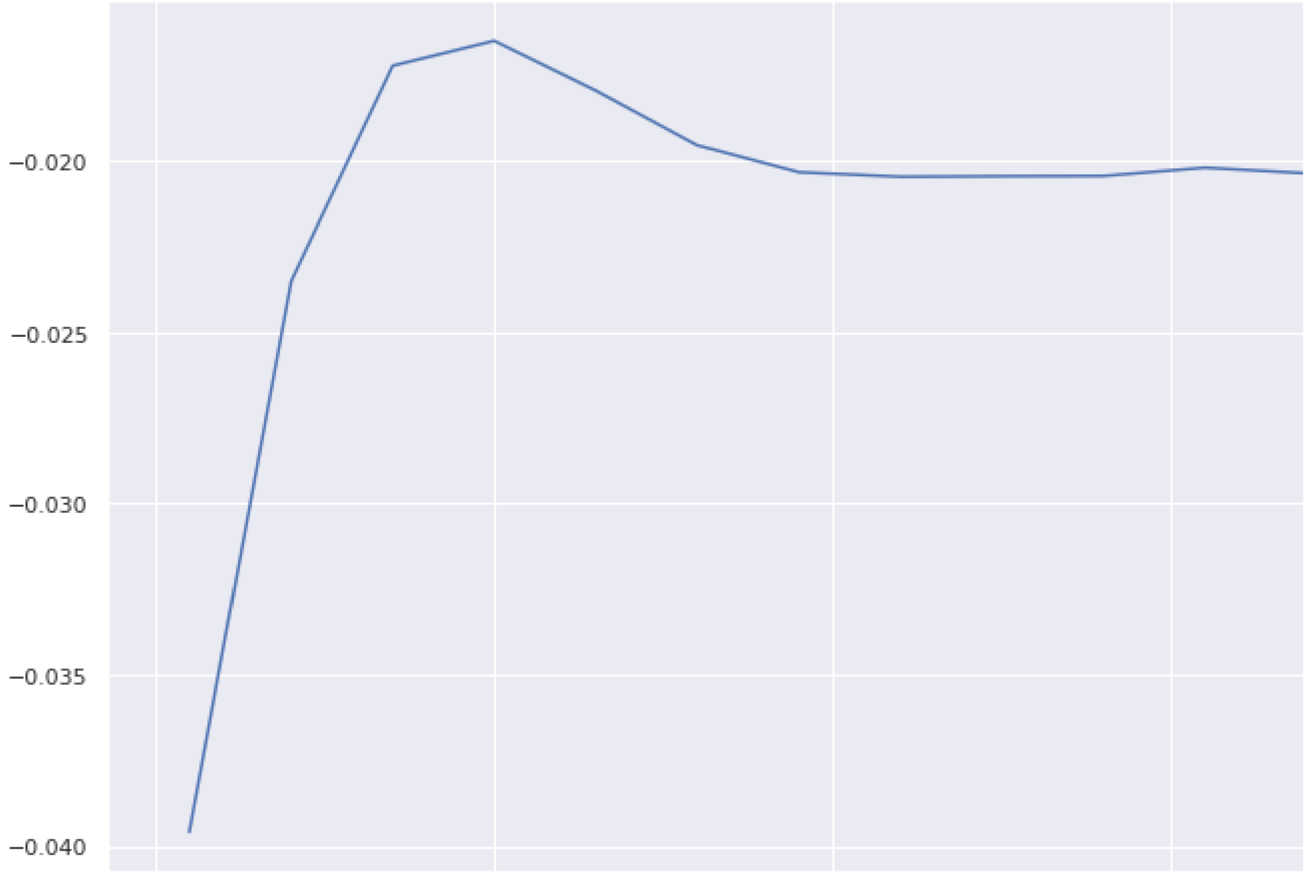
```
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=10,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, presort='deprecated',
                      random_state=None, splitter='best')
```

```
# Лучшее значение параметров
regr_gs_DT.best_params_
```

```
{'max_depth': 10}
```

```
#Изменение качества на тестовой выборке в зависимости от глубины
plt.plot(n_range, regr_gs_DT.cv_results_['mean_test_score'])
```

```
[<matplotlib.lines.Line2D at 0x7ff406a82f28>]
```



SVM

```
n_range1 = np.array(range(1,5,1))
n_range2 = np.array(range(1,10,1))*0.1
tuned_parameters = [{'degree': n_range1}, {'gamma' : n_range2}]
tuned_parameters
```

```
[{'degree': array([1, 2, 3, 4])},
 {'gamma': array([0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9])}]
```

```
regr_gs_SVR = GridSearchCV(SVR(), tuned_parameters, cv=5, scoring='neg_mean_squared_error'
regr_gs_SVR.fit(regr_X_train, regr_Y_train)
```

```
GridSearchCV(cv=5, 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=[{'degree': array([1, 2, 3, 4])},
                         {'gamma': array([0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='neg_mean_squared_error', verbose=0)
```

```
# Лучшая модель
regr_gs_SVR.best_estimator_
```

```
SVR(C=1.0, cache_size=200, coef0=0.0, degree=1, epsilon=0.1, gamma='scale',
    kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False)
```

```
# Лучшее значение параметров
regr_gs_SVR.best_params_
```

```
{'degree': 1}
```

```
#Изменение качества на тестовой выборке в зависимости от глубины
plt.plot(n_range1, regr_gs_SVR.cv_results_)
```

RandomForestRegressor max_depth

```
n_range = np.array(range(1,10,1))
tuned_parameters = [{'max_depth': n_range}]
tuned_parameters
```

```
[{'max_depth': array([1, 2, 3, 4, 5, 6, 7, 8, 9])}]
```

```
regr_gs_RFR = GridSearchCV(RandomForestRegressor(), tuned_parameters, cv=5, scoring='neg_m
regr_gs_RFR.fit(regr_X_train, regr_Y_train)
```

```
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_depth': array([1, 2, 3, 4, 5, 6, 7, 8, 9])}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='neg_mean_squared_error', verbose=0)
```

```
# Лучшая модель
regr_gs_RFR.best_estimator_
```

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=9, 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)
```

```
regr_gs_RFR.best_params_
```

```
{'max_depth': 9}
```

GradientBoostingRegresso

```
n_range = np.array(range(1,10,1))
tuned_parameters = [{'max_depth': n_range}]
tuned_parameters
```

```
[{'max_depth': array([1, 2, 3, 4, 5, 6, 7, 8, 9])}]
```

```
regr_gs_GBR = GridSearchCV(GradientBoostingRegressor(), tuned_parameters, cv=5, scoring='n
regr_gs_GBR.fit(regr_X_train, regr_Y_train)
```

```
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_depth': array([1, 2, 3, 4, 5, 6, 7, 8, 9])}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='neg_mean_squared_error', verbose=0)
```

```
regr_gs_GBR.best_estimator_
```

```
GradientBoostingRegressor(alpha=0.9, ccp_alpha=0.0, criterion='friedman_mse',
                          init=None, learning_rate=0.1, loss='ls', max_depth=9,
                          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)
```

```
regr_gs_GBR.best_params_
```

```
{'max_depth': 9}
```

LinearRegression

```
n_range = np.array(range(1,10,1))*0.1
tuned_parameters = [{'n_jobs': n_range}]
tuned_parameters
```

```
[{'n_jobs': array([0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9])}]
```

```
regr_gs_LR = GridSearchCV(LinearRegression(), tuned_parameters, cv=5, scoring='neg_mean_sq
regr_gs_LR.fit(regr_X_train, regr_Y_train)
```

```
GridSearchCV(cv=5, error_score=nan,
             estimator=LinearRegression(copy_X=True, fit_intercept=True,
                                        n_jobs=None, normalize=False),
             iid='deprecated', n_jobs=None,
             param_grid=[{'n_jobs': array([0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='neg_mean_squared_error', verbose=0)
```

```
regr_gs_LR.best_estimator_
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=0.1, normalize=False)
```

```
regr_gs_LR.best_params_
```

```
{'n_jobs': 0.1}
```

Повторение построения базового решения для задачи регресии

```
regr_models_grid = {'KNN_801':regr_gs_KN.best_estimator_,
                    'LR':regr_gs_LR.best_estimator_,
                    'SVR':regr_gs_SVR.best_estimator_,
                    'RFR':regr_gs_RFR.best_estimator_,
                    'GBR' : regr_gs_GBR.best_estimator_,
                    'LR' : regr_gs_LR.best_estimator_
                    }
```

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

```
*****************************************************
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=11, p=2,
                    weights='uniform')

MAE=0.085, MSE=0.016, R2=0.722
*****************************************************
*****************************************************
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=0.1, normalize=False)

MAE=0.104, MSE=0.02, R2=0.645
*****************************************************
*****************************************************
SVR(C=1.0, cache_size=200, coef0=0.0, degree=1, epsilon=0.1, gamma='scale',
    kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False)

MAE=0.085, MSE=0.015, R2=0.741
*****************************************************
*****************************************************
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=9, 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=0.074, MSE=0.012, R2=0.79
*****************************************************
*****************************************************
```

Решение задачи регрессии

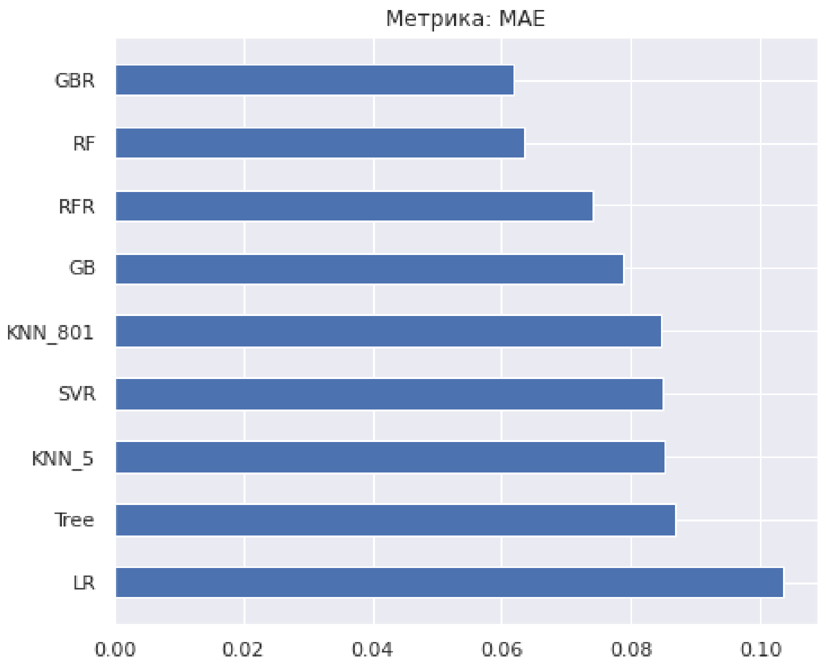
```
                          max_features=None, max_leaf_nodes=None
```

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

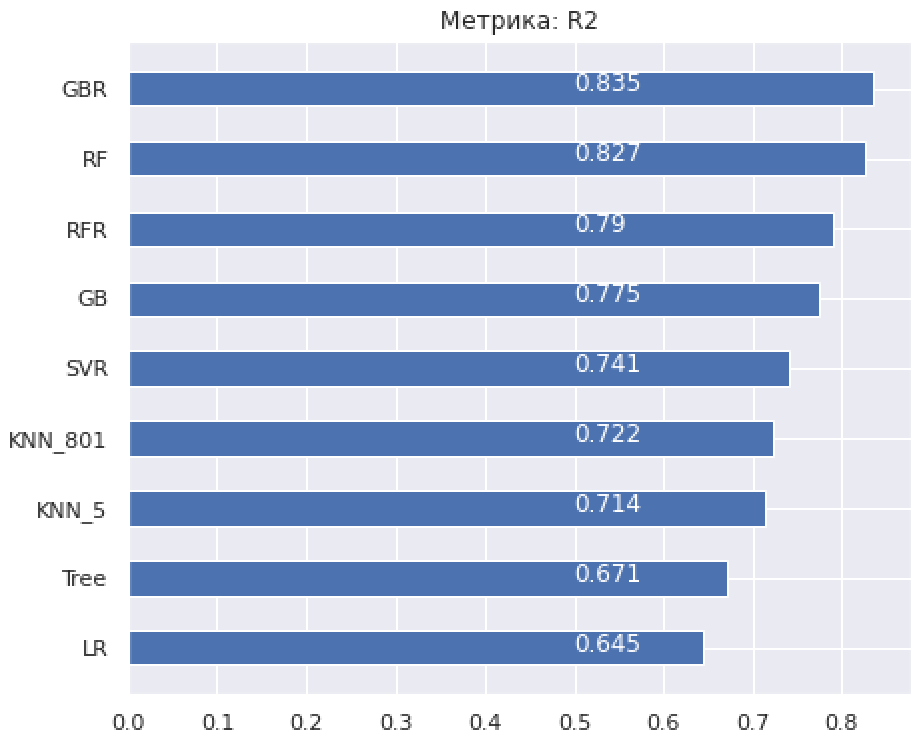
```
array(['MAE', 'MSE', 'R2'], dtype=object)
```

Double-click (or enter) to edit

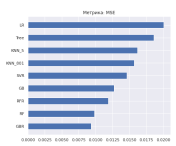
```
regrMetricLogger.plot('Метрика: ' + 'MAE', 'MAE', ascending=False, figsize=(7, 6))
```




```
regrMetricLogger.plot('Метрика: ' + 'R2', 'R2', ascending=True, figsize=(7, 6))
```




```
regrMetricLogger.plot('Метрика: ' + 'MSE', 'MSE', ascending=True, figsize=(7, 6))
```



Как видно из гистограмм наилучшим методом будет линейная регрессия либо градиентны

```
!apt-get install texlive texlive-xetex texlive-latex-extra pandoc
!pip install pypandoc
```

```
Reading package lists... Done
Building dependency tree
Reading state information... Done
pandoc is already the newest version (1.19.2.4~dfsg-1build4).
texlive is already the newest version (2017.20180305-1).
texlive-latex-extra is already the newest version (2017.20180305-2).
texlive-xetex is already the newest version (2017.20180305-1).
0 upgraded, 0 newly installed, 0 to remove and 31 not upgraded.
Requirement already satisfied: pypandoc in /usr/local/lib/python3.6/dist-packages (1.
Requirement already satisfied: wheel>=0.25.0 in /usr/local/lib/python3.6/dist-package
Requirement already satisfied: setuptools in /usr/local/lib/python3.6/dist-packages (
Requirement already satisfied: pip>=8.1.0 in /usr/local/lib/python3.6/dist-packages (
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.m
```

```
cp "/content/drive/My Drive/Colab Notebooks/Untitled.ipynb" ./
```

```
cp: cannot stat '/content/drive/My Drive/Colab Notebooks/Untitled.ipynb': No such fil
```