```python
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

data = pd.read_csv("kc_house_data.csv")
data.head()
data.shape
```

```
(21613, 21)
```

```python
data = data.drop(columns = ["id","date","zipcode"])#3 столбца можно удалить тк они не несут информации о самом доме
data = data[0:10000]
data.isna().sum()# пропусков в данных нет
```

```
price            0
bedrooms         0
bathrooms        0
sqft_living      0
sqft_lot         0
floors           0
waterfront       0
view             0
condition        0
grade            0
sqft_above       0
sqft_basement    0
yr_built         0
yr_renovated     0
lat              0
long             0
sqft_living15    0
sqft_lot15       0
dtype: int64
```

```python
y = data["grade"]# выделяем определяемый столбец
data = data.drop(columns = "grade") # удаляем его из данных
data.head()
data.dtypes# все переменные числовые => отдельно кодировать что-то не прийдется
```

```
price            float64
bedrooms           int64
bathrooms        float64
sqft_living        int64
sqft_lot           int64
floors           float64
waterfront         int64
view               int64
condition          int64
sqft_above         int64
```

```
sqft_basement      int64
yr_built           int64
yr_renovated       int64
lat              float64
long             float64
sqft_living15      int64
sqft_lot15         int64
dtype: object
```

```
from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
data = pd.DataFrame(ss.fit_transform(data), columns = data.columns)#
нормализуем данные
data.head()
data.shape
```

```
(10000, 17)
```

```
from  sklearn.model_selection import train_test_split
y = y.apply(lambda x: 0 if x < 7.5 else 1) # закодируем относительно
условия переменную grade
data_train, data_test, y_train, y_test = train_test_split(data, y,
test_size = 0.3, random_state = 123)
#поделим данные обучающую и тестовую выборки
y_train.head()
data_train.head()
```

```
         price  bedrooms  bathrooms  sqft_living  sqft_lot
floors  \
3144 -0.552869  0.696948  -0.406653    -0.749663 -0.134236 -0.844194

9939  0.956965  2.880367   0.571426     0.841743 -0.165164 -0.844194

7925  0.009789  0.696948  -0.406653    -0.519184 -0.275402 -0.844194

309   0.349507  0.696948   0.897453     1.390504 -0.121336  1.109507

9415  0.682590  0.696948   0.897453     1.423430 -0.235858  1.109507


      waterfront      view  condition  sqft_above  sqft_basement
yr_built  \
3144   -0.089235 -0.311479  -0.669848   -0.468145      -0.673136 -
1.760696
9939   -0.089235 -0.311479   2.331936   -0.394148       2.410152 -
0.009876
7925   -0.089235 -0.311479   0.831044   -1.183456       1.079236 -
2.046544
309    -0.089235 -0.311479   2.331936    0.580154       1.766876 -
0.259993
9415   -0.089235 -0.311479  -0.669848    1.973776      -0.673136
```

```
1.669482
```

	yr_renovated	lat	long	sqft_living15	sqft_lot15
3144	-0.219329	1.409234	-0.096317	-0.484624	-0.064787
9939	-0.219329	0.036939	0.507951	0.140191	-0.163342
7925	-0.219329	0.854461	-1.125011	-1.168945	-0.355986
309	-0.219329	-0.158696	0.248979	1.107167	-0.184210
9415	-0.219329	0.893017	1.428741	2.453495	-0.282802

```
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score, precision_score, f1_score,
recall_score
from sklearn.model_selection import GridSearchCV
dtc = DecisionTreeClassifier()
pg = {'max_depth': [1,5,10,15,20,25]}
gs = GridSearchCV(estimator=dtc, param_grid = pg)
gs.fit(data_train, y_train)
dtc_best = gs.best_estimator_
prediction = dtc_best.predict(data_test)
gs.best_params_ #лучшая max_depth = 10
```

```
{'max_depth': 5}
```

```
precision_score(y_test, prediction)
```

```
0.8453441295546559
```

```
f1_score(y_test,prediction)
```

```
0.8201099764336214
```

```
accuracy_score(y_test,prediction)
```

```
0.8473333333333334
```

```
recall_score(y_test, prediction)
```

```
0.7963386727688787
```

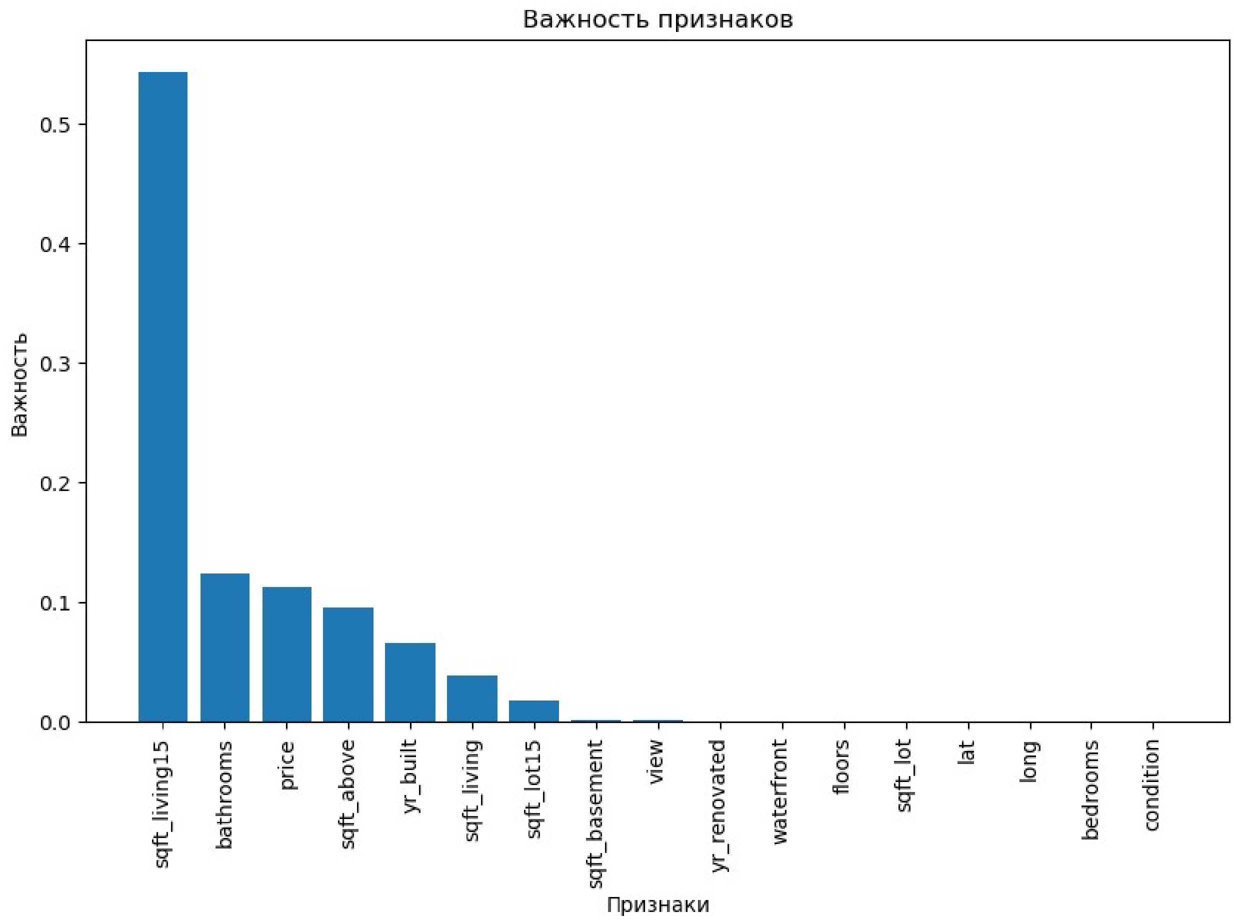
```
# доля правильных ответов(accuracy)
# точность предсказания, сколько процентов предположительного
положительного класса, действительно таковыми являются(precision)
# сколько процентов положительного класса находит модель (recall)
# гармоническое средние recall и precision (f1)
# классификатор показывает хорошие показтели по всем метрикам

# отображение важности признаков для классификации

importances = dtc_best.feature_importances_
names = data_train.columns.tolist()
plt.figure(figsize=(10, 6))
plt.title("Важность признаков")
```

```
plt.xlabel("Признаки")
plt.ylabel("Важность")
indices = np.argsort(importances)[::-1]
plt.bar(range(data_train.shape[1]), importances[indices],
align='center')
plt.xticks(range(data_train.shape[1]), [names[i] for i in indices],
rotation=90)
```

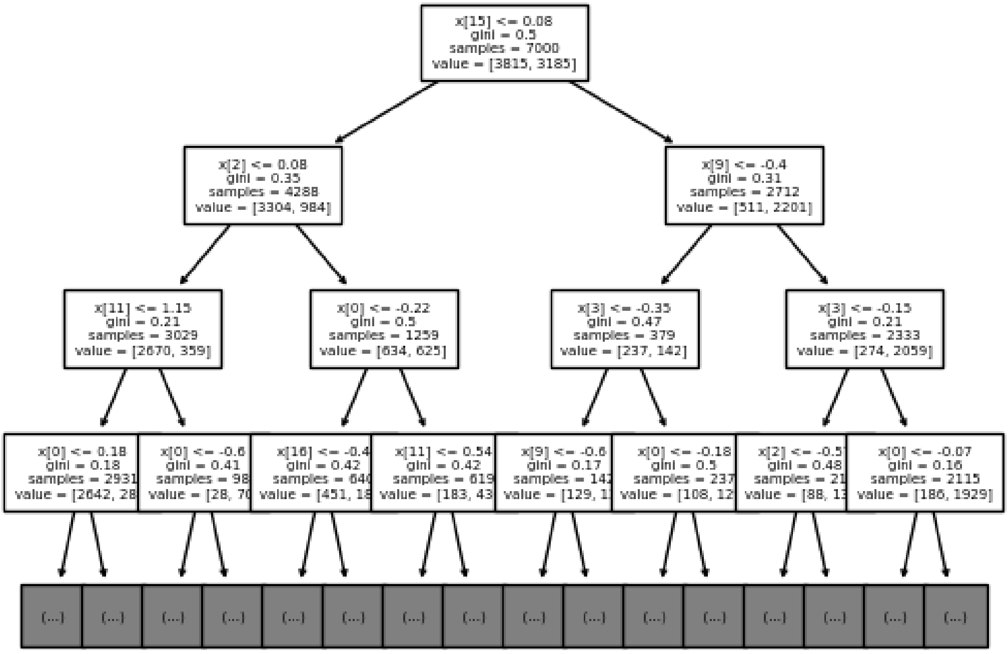
```
([<matplotlib.axis.XTick at 0x15cf6586c90>,
  <matplotlib.axis.XTick at 0x15cfbb1b850>,
  <matplotlib.axis.XTick at 0x15cf6541050>,
  <matplotlib.axis.XTick at 0x15cf65dd750>,
  <matplotlib.axis.XTick at 0x15cf65dfa50>,
  <matplotlib.axis.XTick at 0x15cf65e5d10>,
  <matplotlib.axis.XTick at 0x15cf65e4950>,
  <matplotlib.axis.XTick at 0x15cf65f14d0>,
  <matplotlib.axis.XTick at 0x15cf65f3810>,
  <matplotlib.axis.XTick at 0x15cf65f9b50>,
  <matplotlib.axis.XTick at 0x15cf65fbcd0>,
  <matplotlib.axis.XTick at 0x15cf65f9290>,
  <matplotlib.axis.XTick at 0x15cf66028d0>,
  <matplotlib.axis.XTick at 0x15cf6604b90>,
  <matplotlib.axis.XTick at 0x15cf6606e90>,
  <matplotlib.axis.XTick at 0x15cf660d210>,
  <matplotlib.axis.XTick at 0x15cf6607d50>],
 [Text(0, 0, 'sqft_living15'),
  Text(1, 0, 'bathrooms'),
  Text(2, 0, 'price'),
  Text(3, 0, 'sqft_above'),
  Text(4, 0, 'yr_built'),
  Text(5, 0, 'sqft_living'),
  Text(6, 0, 'sqft_lot15'),
  Text(7, 0, 'sqft_basement'),
  Text(8, 0, 'view'),
  Text(9, 0, 'yr_renovated'),
  Text(10, 0, 'waterfront'),
  Text(11, 0, 'floors'),
  Text(12, 0, 'sqft_lot'),
  Text(13, 0, 'lat'),
  Text(14, 0, 'long'),
  Text(15, 0, 'bedrooms'),
  Text(16, 0, 'condition')])
```




```python
#отображение усеченного дерева
from sklearn import tree
tree.plot_tree(dtc_best, max_depth = 3, fontsize = 5, precision = 2)
```

```
[Text(0.5, 0.9, 'x[15] <= 0.08\ngini = 0.5\nsamples = 7000\nvalue =
[3815, 3185]'),
 Text(0.25, 0.7, 'x[2] <= 0.08\ngini = 0.35\nsamples = 4288\nvalue =
[3304, 984]'),
 Text(0.125, 0.5, 'x[11] <= 1.15\ngini = 0.21\nsamples = 3029\nvalue =
[2670, 359]'),
 Text(0.0625, 0.3, 'x[0] <= 0.18\ngini = 0.18\nsamples = 2931\nvalue =
[2642, 289]'),
 Text(0.03125, 0.1, '\n  (...)  \n'),
 Text(0.09375, 0.1, '\n  (...)  \n'),
 Text(0.1875, 0.3, 'x[0] <= -0.6\ngini = 0.41\nsamples = 98\nvalue =
[28, 70]'),
 Text(0.15625, 0.1, '\n  (...)  \n'),
 Text(0.21875, 0.1, '\n  (...)  \n'),
 Text(0.375, 0.5, 'x[0] <= -0.22\ngini = 0.5\nsamples = 1259\nvalue =
[634, 625]'),
 Text(0.3125, 0.3, 'x[16] <= -0.4\ngini = 0.42\nsamples = 640\nvalue =
```

```
[451, 189]'),
 Text(0.28125, 0.1, '\n  (...)  \n'),
 Text(0.34375, 0.1, '\n  (...)  \n'),
 Text(0.4375, 0.3, 'x[11] <= 0.54\ngini = 0.42\nsamples = 619\nvalue =
[183, 436]'),
 Text(0.40625, 0.1, '\n  (...)  \n'),
 Text(0.46875, 0.1, '\n  (...)  \n'),
 Text(0.75, 0.7, 'x[9] <= -0.4\ngini = 0.31\nsamples = 2712\nvalue =
[511, 2201]'),
 Text(0.625, 0.5, 'x[3] <= -0.35\ngini = 0.47\nsamples = 379\nvalue =
[237, 142]'),
 Text(0.5625, 0.3, 'x[9] <= -0.6\ngini = 0.17\nsamples = 142\nvalue =
[129, 13]'),
 Text(0.53125, 0.1, '\n  (...)  \n'),
 Text(0.59375, 0.1, '\n  (...)  \n'),
 Text(0.6875, 0.3, 'x[0] <= -0.18\ngini = 0.5\nsamples = 237\nvalue =
[108, 129]'),
 Text(0.65625, 0.1, '\n  (...)  \n'),
 Text(0.71875, 0.1, '\n  (...)  \n'),
 Text(0.875, 0.5, 'x[3] <= -0.15\ngini = 0.21\nsamples = 2333\nvalue =
[274, 2059]'),
 Text(0.8125, 0.3, 'x[2] <= -0.57\ngini = 0.48\nsamples = 218\nvalue =
[88, 130]'),
 Text(0.78125, 0.1, '\n  (...)  \n'),
 Text(0.84375, 0.1, '\n  (...)  \n'),
 Text(0.9375, 0.3, 'x[0] <= -0.07\ngini = 0.16\nsamples = 2115\nvalue
= [186, 1929]'),
 Text(0.90625, 0.1, '\n  (...)  \n'),
 Text(0.96875, 0.1, '\n  (...)  \n')]
```




```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier()# построим классификатор типа
RandomForestClassifier
rf.fit(data_train, y_train)
prediction = rf.predict(data_test)

precision_score(y_test, prediction)

0.8528735632183908

f1_score(y_test,prediction)

0.8509174311926605

accuracy_score(y_test,prediction)

0.87

recall_score(y_test, prediction)

0.8489702517162472

# классификатор показывает хорошие показтели по всем метрикам, лучше
чем DecisionTreeClassifier

from sklearn.model_selection import GridSearchCV
rf1 = RandomForestClassifier()
pg1 = {'n_estimators': range(100,300,50), 'max_depth': [10, 30, 50],
'min_samples_split': [2, 4, 6], 'min_samples_leaf': [1, 2, 4]}
```

```
gs1 = GridSearchCV(estimator=rf1, param_grid = pg1)
gs1.fit(data_train, y_train)
gs1.best_params_
rf_best1 = gs1.best_estimator_
prediction = rf_best1.predict(data_test)
gs1.best_params_
```

```
{'max_depth': 50,
 'min_samples_leaf': 2,
 'min_samples_split': 6,
 'n_estimators': 150}
```

```
precision_score(y_test, prediction)
```

```
0.8571428571428571
```

```
f1_score(y_test,prediction)
```

```
0.8541905855338692
```

```
accuracy_score(y_test,prediction)
```

```
0.873
```

```
recall_score(y_test, prediction)
```

```
0.851258581235698
```

```
# классификатор показывает хорошие показтели по всем метрикам,
изменнное параметров леса делает классификатор еще лучше
```

```
rf2 = RandomForestClassifier()
pg2 = {'n_estimators': range(100,200,10), 'max_depth':
[gs1.best_params_['max_depth']], 'min_samples_split':
[gs1.best_params_['min_samples_split']], 'min_samples_leaf':
[gs1.best_params_['min_samples_leaf']]}
gs2 = GridSearchCV(estimator=rf2, param_grid = pg2)
gs2.fit(data_train, y_train)
rf_best2 = gs2.best_estimator_
prediction = rf_best2.predict(data_test)
gs2.best_params_
```

```
{'max_depth': 50,
 'min_samples_leaf': 2,
 'min_samples_split': 6,
 'n_estimators': 160}
```

```
precision_score(y_test, prediction)
```

```
0.8626247122026094
```

```
f1_score(y_test,prediction)
```

```
0.8599846977811783
```

```
accuracy_score(y_test,prediction)
```

```
0.878
```

```
recall_score(y_test, prediction)
```

```
0.8573607932875668
```

```
# классификатор показывает хорошие показтели по всем метрикам, найдено
самое лучшее колличество деревьев по шагу 10 - 160, при n_estimators =
160 классификатор получается самый точный
```