



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»

ТМО РК2

Горкунов Н.М. ИУ5-63Б  
5 июня 2024 г.

# 1 ТМО РК2 ИУ5-63Б Горкунов Николай

## 2 Задание.

2.1 Для заданного набора данных (№3) постройте модели классификации или регрессии (регрессии для MEDV). Для построения моделей используйте методы 1 и 2 (дерево решений и случайный лес). Оцените качество моделей на основе подходящих метрик качества (не менее двух метрик). Какие метрики качества Вы использовали и почему? Какие выводы Вы можете сделать о качестве построенных моделей? Для построения моделей необходимо выполнить требуемую предобработку данных: заполнение пропусков, кодирование категориальных признаков, и т.д.

## 3 Набор данных №3.

### 3.1 Boston housing dataset

```
[1]: import warnings
warnings.filterwarnings("ignore")
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error
import seaborn as sns
import time
import matplotlib.pyplot as plt
from kaggle.api.kaggle_api_extended import KaggleApi
pd.options.display.max_columns = None
```







```
[2]: kaggle_api = KaggleApi()
kaggle_api.authenticate()
kaggle_api.dataset_download_files('altavish/boston-housing-dataset',
↳ unzip=True)
```

Dataset URL: [https://www.kaggle.com/datasets/altavish/](https://www.kaggle.com/datasets/altavish/boston-housing-dataset)  
↳ boston-housing-dataset

### 3.2 Смотрю, что в данных

```
[3]: df = pd.read_csv('HousingData.csv')
      print(df.shape)
      df.head()
```

(506, 14)

```
[3]:      CRIM      ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD  TAX  
      ↳PTRATIO \
0  0.00632  18.0    2.31    0.0  0.538   6.575   65.2   4.0900    1   296  
      ↳15.3
1  0.02731   0.0    7.07    0.0  0.469   6.421   78.9   4.9671    2   242  
      ↳17.8
2  0.02729   0.0    7.07    0.0  0.469   7.185   61.1   4.9671    2   242  
      ↳17.8
3  0.03237   0.0    2.18    0.0  0.458   6.998   45.8   6.0622    3   222  
      ↳18.7
4  0.06905   0.0    2.18    0.0  0.458   7.147   54.2   6.0622    3   222  
      ↳18.7

      B  LSTAT  MEDV
0  396.90   4.98  24.0
1  396.90   9.14  21.6
2  392.83   4.03  34.7
3  394.63   2.94  33.4
4  396.90   NaN  36.2
```

### 3.3 Проверяю типы данных

```
[4]: df.dtypes
```

```
[4]: CRIM      float64
      ZN       float64
      INDUS   float64
      CHAS    float64
      NOX     float64
      RM      float64
      AGE     float64
      DIS     float64
      RAD     int64
```

```
TAX          int64
PTRATIO      float64
B            float64
LSTAT        float64
MEDV         float64
dtype: object
```

### 3.4 Проверяю значения категориальных признаков

```
[5]: df.CHAS.unique()
```

```
[5]: array([ 0., nan,  1.])
```

### 3.5 Проверяю пропуски

```
[6]: df.isna().sum()
```

```
[6]: CRIM      20
     ZN        20
     INDUS    20
     CHAS      20
     NOX       0
     RM        0
     AGE      20
     DIS       0
     RAD       0
     TAX       0
     PTRATIO   0
     B         0
     LSTAT     20
     MEDV      0
     dtype: int64
```

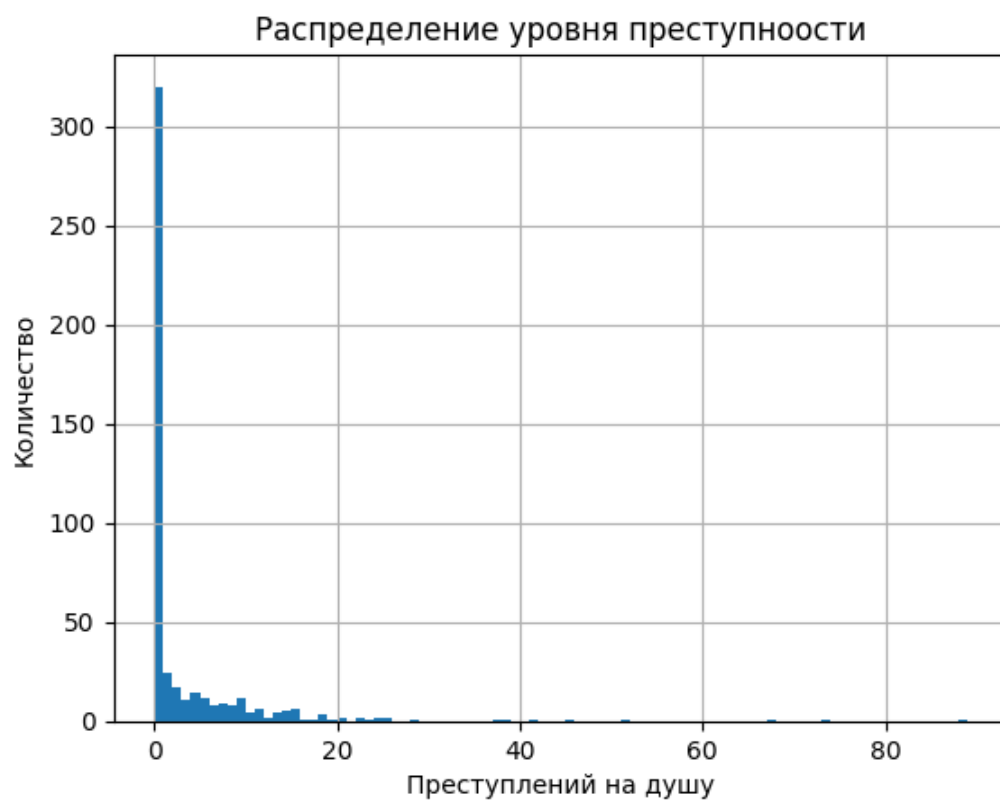
### 3.6 Заполняю пропуски в численном признаке “CRIM” в соответствии с описанием “CRIM - per capita crime rate by town”

```
[7]: df[df.CRIM == 0]
```

```
[7]: Empty DataFrame
```

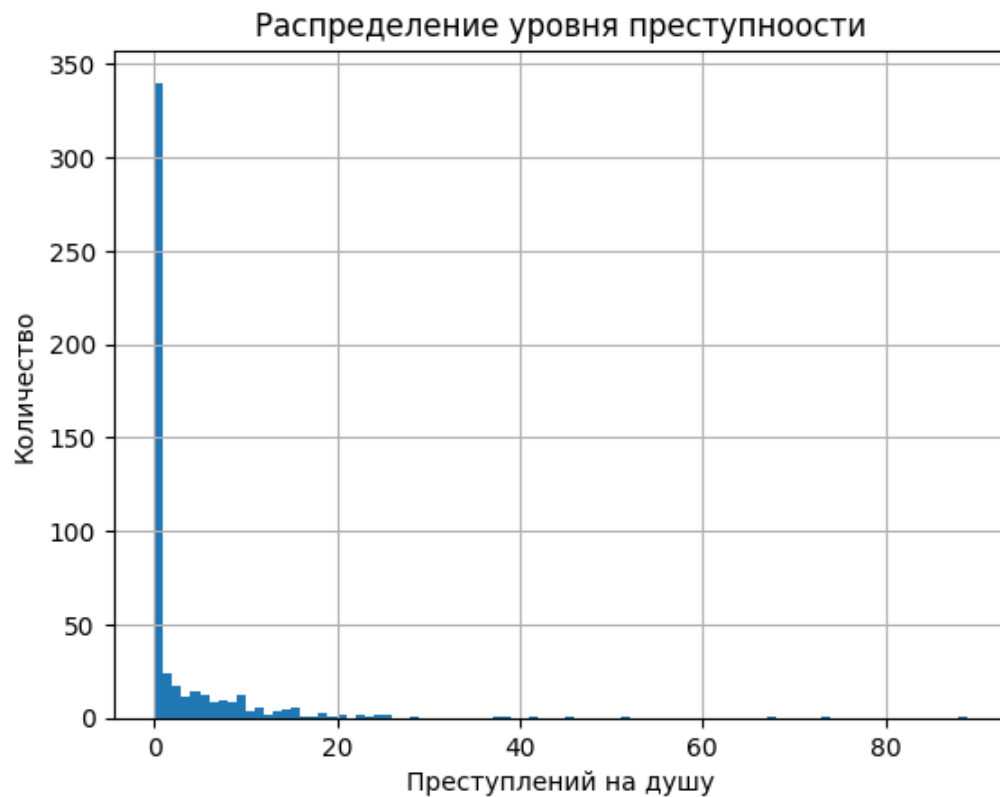
```
Columns: [CRIM, ZN, INDUS, CHAS, NOX, RM, AGE, DIS, RAD, TAX, PTRATIO, B, LSTAT,  
→MEDV]  
Index: []
```

```
[8]: df.CRIM.hist(bins=range(90))  
plt.title('Распределение уровня преступности')  
plt.xlabel('Преступлений на душу')  
plt.ylabel('Количество')  
plt.show()
```



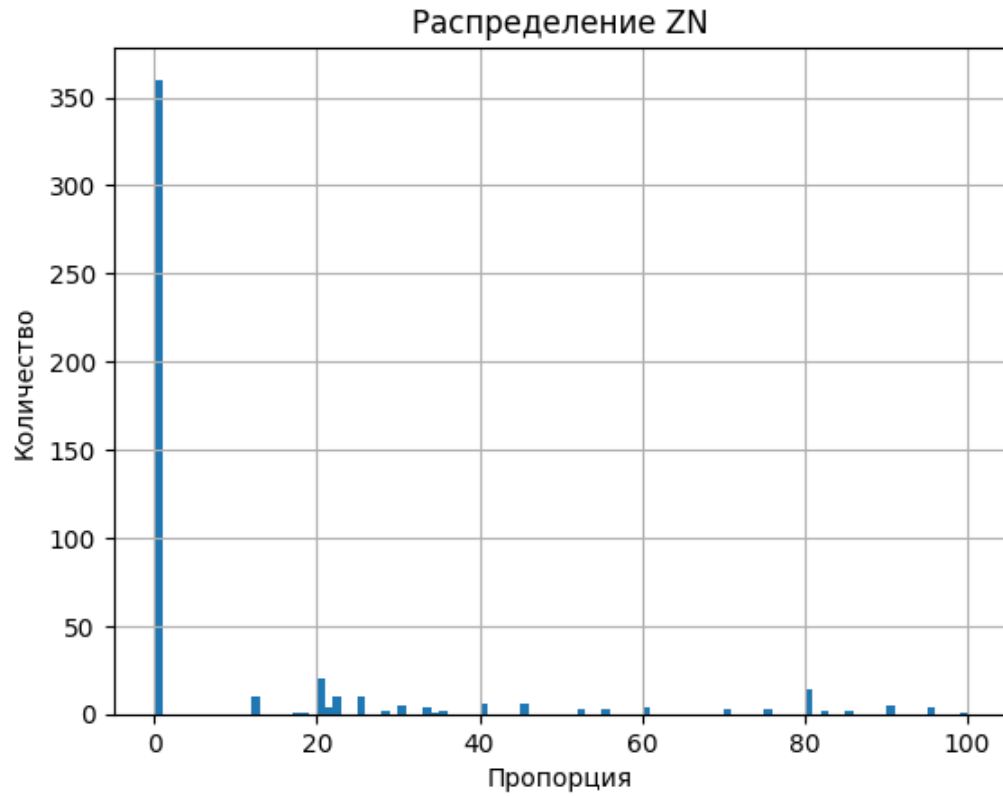
```
[9]: df = df.fillna(value={"CRIM": 0})  
  
df.CRIM.hist(bins=range(90))
```

```
plt.title('Распределение уровня преступности')
plt.xlabel('Преступлений на душу')
plt.ylabel('Количество')
plt.show()
```



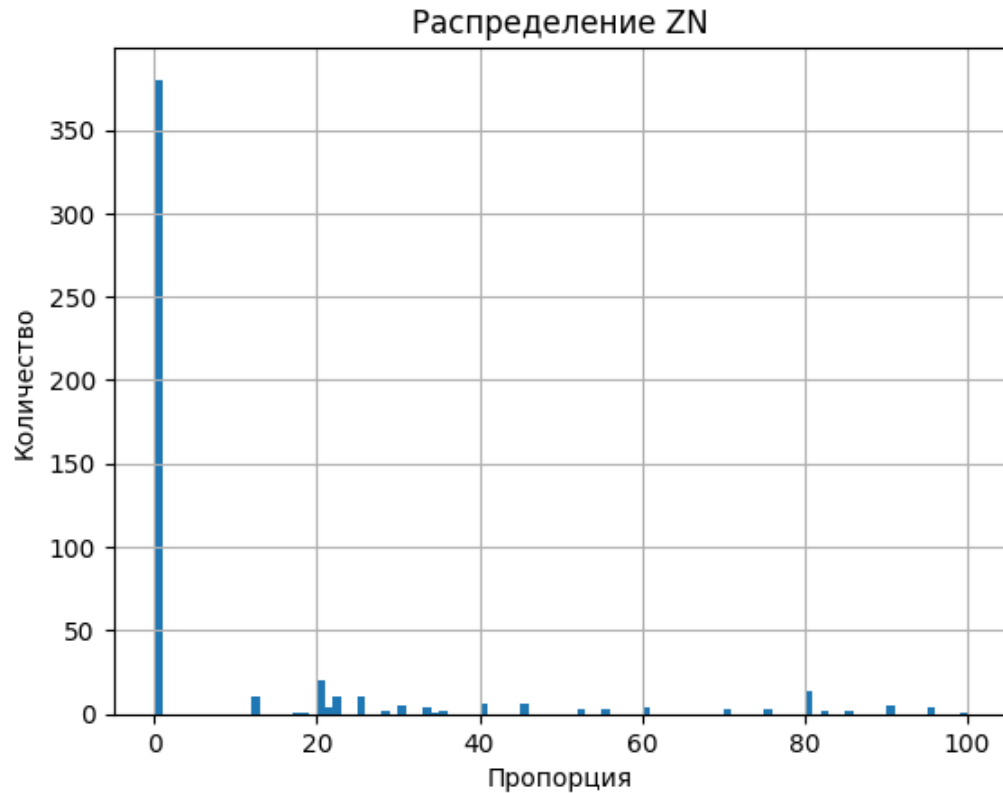
**3.7** Заполняю пропуски в численном признаке “ZN” в соответствии с описанием “ZN - proportion of residential land zoned for lots over 25,000 sq.ft.”

```
[10]: df.ZN.hist(bins=range(101))
plt.title('Распределение ZN')
plt.xlabel('Пропорция')
plt.ylabel('Количество')
plt.show()
```



```
[11]: df = df.fillna(value={"ZN": 0})
```

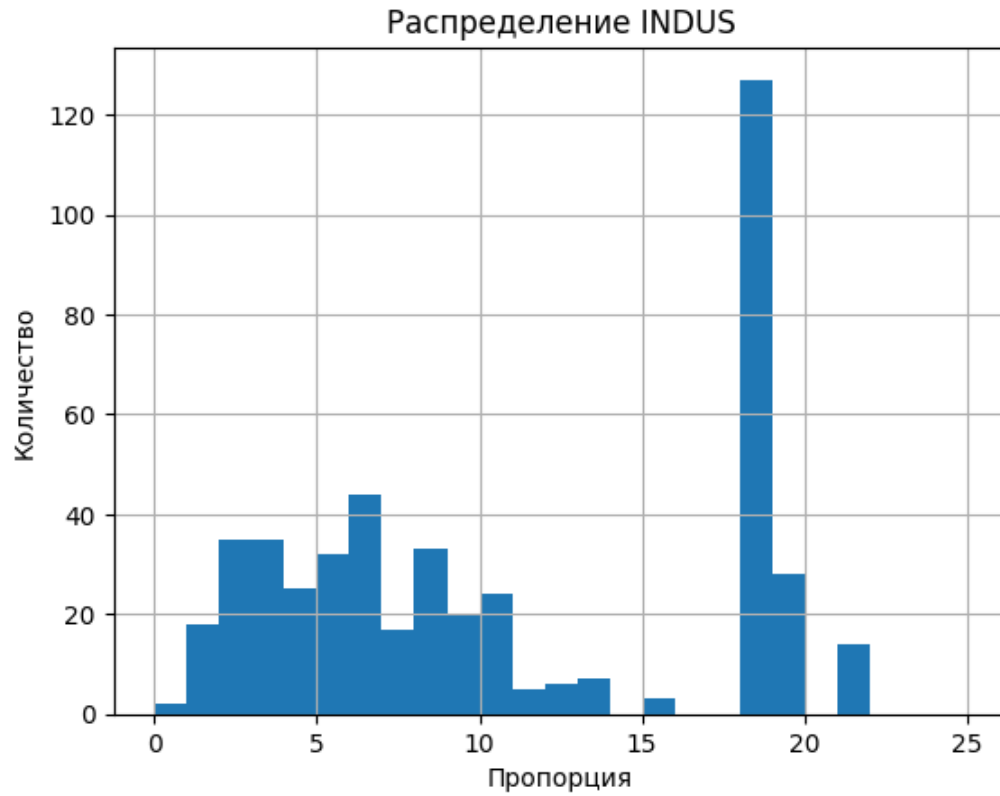
```
df.ZN.hist(bins=range(101))  
plt.title('Распределение ZN')  
plt.xlabel('Пропорция')  
plt.ylabel('Количество')  
plt.show()
```



3.8 Заполняю пропуски в численном признаке “INDUS” в соответствии с описанием “INDUS - proportion of non-retail business acres per town.”

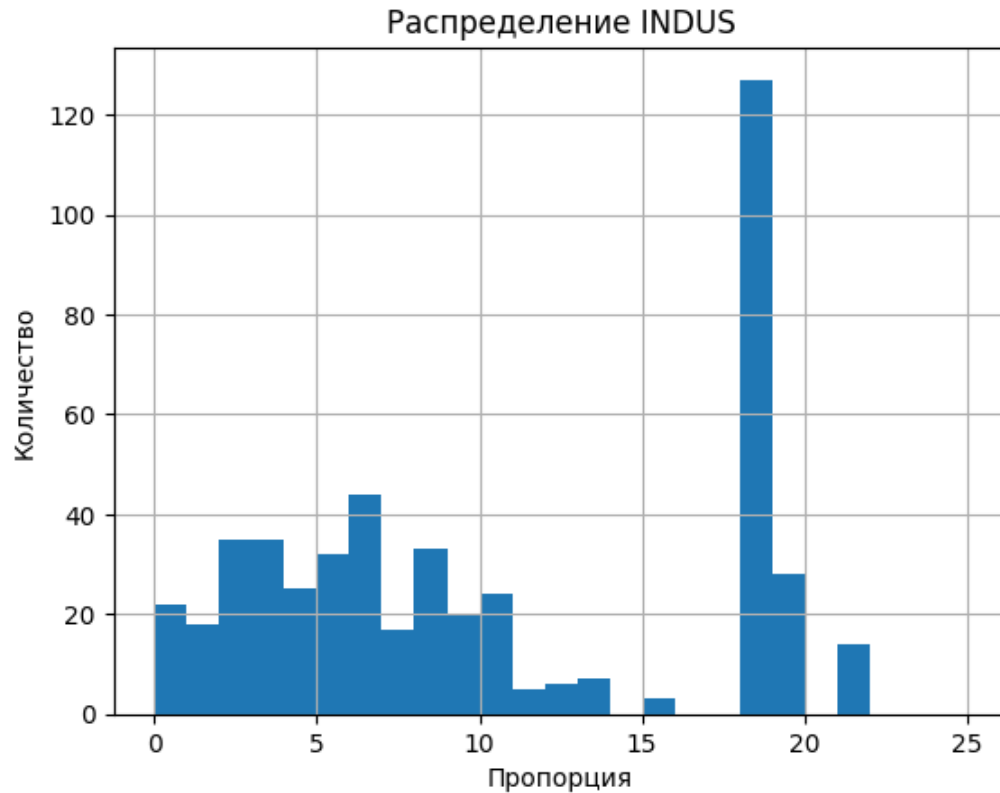
```
[12]: df.INDUS.hist(bins=range(26))
plt.title('Распределение INDUS')
plt.xlabel('Пропорция')
plt.ylabel('Количество')
plt.show()
```





```
[13]: df = df.fillna(value={"INDUS": 0})
```

```
df.INDUS.hist(bins=range(26))  
plt.title('Распределение INDUS')  
plt.xlabel('Пропорция')  
plt.ylabel('Количество')  
plt.show()
```



**3.9 Удаляю пропуски в категориальном признаке “CHAS” в соответствии с описанием “CHAS - Charles River dummy variable (1 if tract bounds river; 0 otherwise)”**

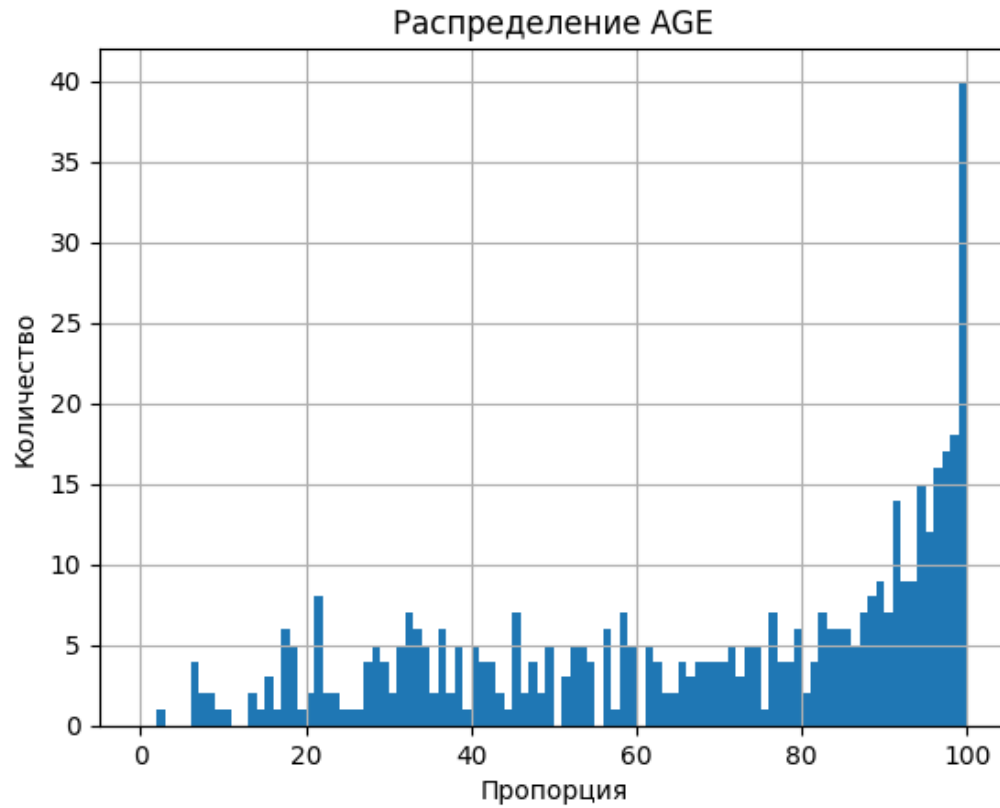
```
[14]: df = df.dropna(subset=['CHAS'])
      df.isna().sum()
```

```
[14]: CRIM      0
      ZN        0
      INDUS    0
      CHAS      0
      NOX       0
      RM        0
      AGE      20
```

```
DIS          0
RAD          0
TAX          0
PTRATIO      0
B            0
LSTAT        19
MEDV         0
dtype: int64
```

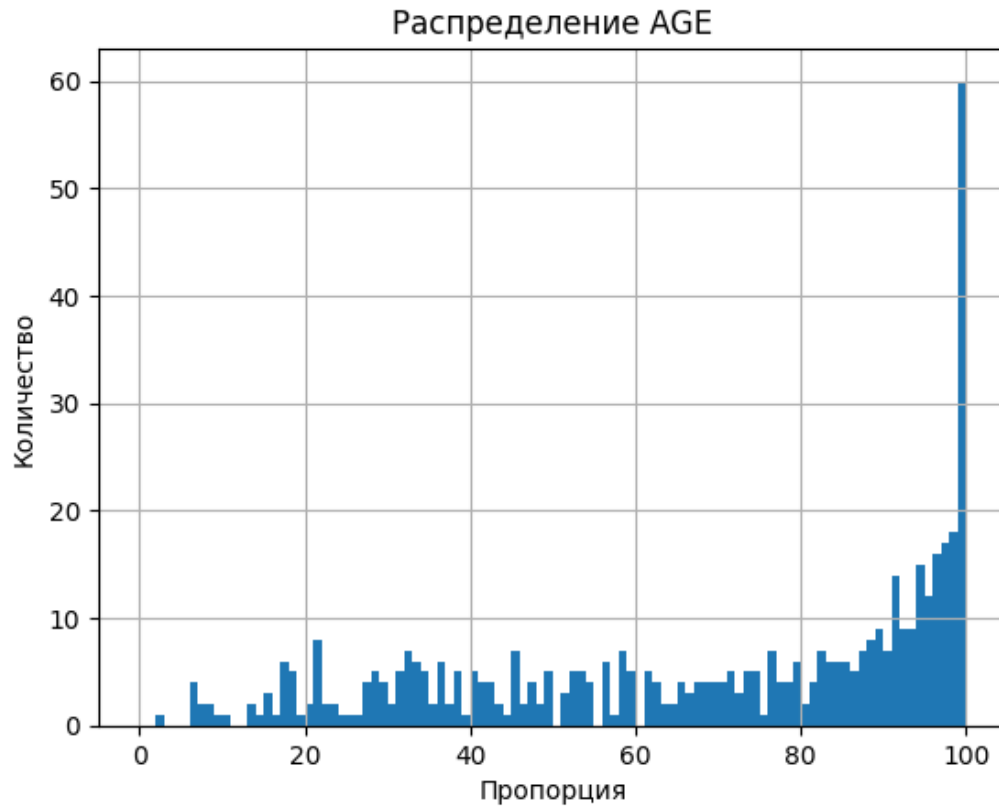
### 3.10 Заполняю пропуски в численном признаке “AGE” в соответствии с описанием “AGE - proportion of owner-occupied units built prior to 1940”

```
[15]: df.AGE.hist(bins=range(101))
plt.title('Распределение AGE')
plt.xlabel('Пропорция')
plt.ylabel('Количество')
plt.show()
```



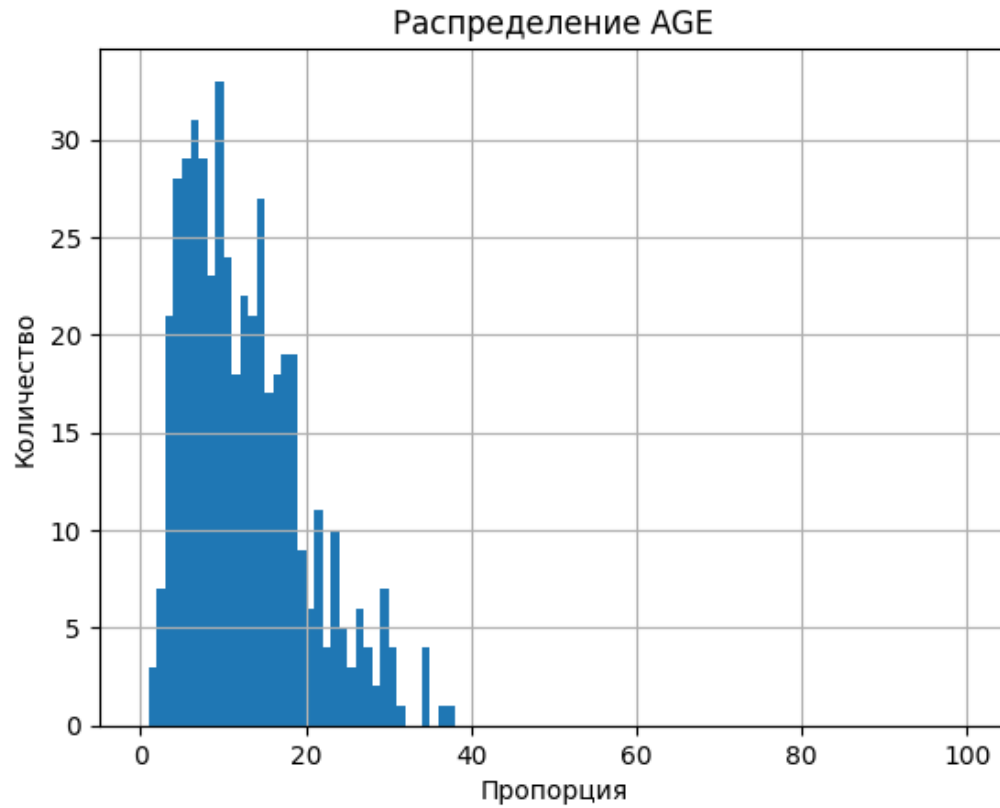
```
[16]: df = df.fillna(value={"AGE": 100})
```

```
df.AGE.hist(bins=range(101))  
plt.title('Распределение AGE')  
plt.xlabel('Пропорция')  
plt.ylabel('Количество')  
plt.show()
```



### 3.11 Заполняю пропуски в численном признаке “LSTAT” в соответствии с описанием “LSTAT - % lower status of the population”

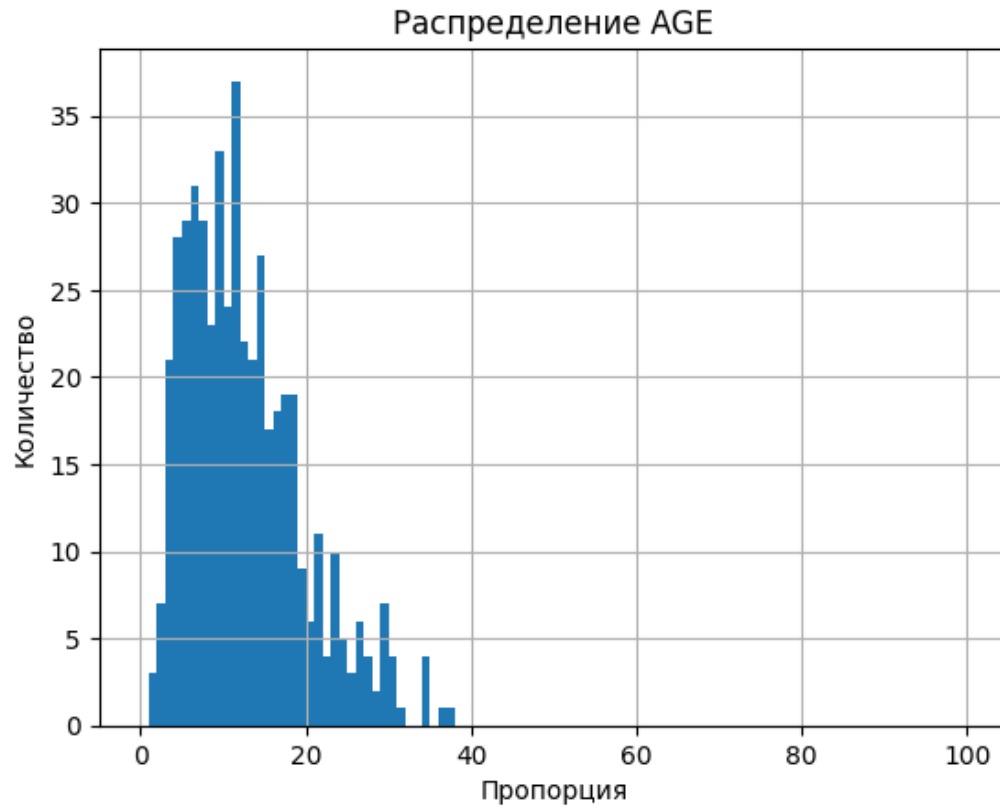
```
[17]: df.LSTAT.hist(bins=range(101))  
plt.title('Распределение AGE')  
plt.xlabel('Пропорция')  
plt.ylabel('Количество')  
plt.show()
```



```
[18]: med = df.LSTAT.median()
print(med)
df = df.fillna(value={"LSTAT": int(med)})

df.LSTAT.hist(bins=range(101))
plt.title('Распределение AGE')
plt.xlabel('Пропорция')
plt.ylabel('Количество')
plt.show()
```

11.32



```
[19]: df.isna().sum()
```

```
[19]: CRIM      0
      ZN       0
      INDUS   0
      CHAS    0
      NOX     0
      RM      0
      AGE     0
      DIS     0
      RAD     0
      TAX     0
      PTRATIO 0
      B       0
```

```
LSTAT      0
MEDV       0
dtype: int64
```

### 3.12 Преобразую категориальные признаки (one hot encoding)

```
[20]: for to_enc in ["CHAS"]:
        one_hot = pd.get_dummies(df[to_enc]).astype(int)
        del df[to_enc]
        df = df.join(one_hot)
df.columns = df.columns.map(str)
df.head()
```

```
[20]:      CRIM      ZN  INDUS      NOX      RM      AGE      DIS  RAD  TAX  PTRATIO  \
0  0.00632  18.0    2.31  0.538  6.575  65.2  4.0900    1  296    15.3
1  0.02731   0.0    7.07  0.469  6.421  78.9  4.9671    2  242    17.8
2  0.02729   0.0    7.07  0.469  7.185  61.1  4.9671    2  242    17.8
3  0.03237   0.0    2.18  0.458  6.998  45.8  6.0622    3  222    18.7
4  0.06905   0.0    2.18  0.458  7.147  54.2  6.0622    3  222    18.7
```

```
      B  LSTAT  MEDV  0.0  1.0
0  396.90   4.98  24.0    1    0
1  396.90   9.14  21.6    1    0
2  392.83   4.03  34.7    1    0
3  394.63   2.94  33.4    1    0
4  396.90  11.00  36.2    1    0
```

### 3.13 Провожу разделение на тестовую и обучающую выборки, обучаю и тестирую дерево решений и случайный лес, оцениваю помощью MAE, MSE

```
[21]: def exec_time(start, end):
        diff_time = end - start
        m, s = divmod(diff_time, 60)
        h, m = divmod(m, 60)
        s, m, h = int(round(s, 0)), int(round(m, 0)), int(round(h, 0))
        return("{0:02d}:{1:02d}:{2:02d}".format(h, m, s))
```

```
[22]: y = df.MEDV.copy()
X = df.loc[:, df.columns != "MEDV"].copy()
```



```
[23]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
↳ random_state=42)
```

```
[24]: model = DecisionTreeRegressor(max_depth=7, random_state=42)
```

```
start = time.time()
model.fit(X_train, y_train)
end = time.time()
fitTime = exec_time(start, end)

start = time.time()
y_pred = model.predict(X_test)
end = time.time()
testTime = exec_time(start, end)

start = time.time()
y_train_pred = model.predict(X_train)
end = time.time()
trainTime = exec_time(start, end)

testMAE = mean_absolute_error(y_test, y_pred)
trainMAE = mean_absolute_error(y_train, y_train_pred)
testMSE = mean_squared_error(y_test, y_pred)
trainMSE = mean_squared_error(y_train, y_train_pred)
print("Test MAE = %.4f" % testMAE)
print("Train MAE = %.4f" % trainMAE)
print("Test MAE = %.4f" % testMSE)
print("Train MAE = %.4f" % trainMSE)
```

```
Test MAE = 3.0585
Train MAE = 1.1531
Test MAE = 21.4773
Train MAE = 2.7920
```

```
[25]: DecisionTreeRegressorMAE = pd.DataFrame({
    "Train MAE" : [trainMAE],
    "Test MAE" : [testMAE],
    "Train MSE" : [trainMSE],
    "Test MSE" : [testMSE],
    "Fit time" : [fitTime],
    "Test time on train df" : [trainTime],
```

```

    "Test time on test df" : [testTime],
}, index=["DecisionTreeRegressor"])
DecisionTreeRegressorMAE

```

```

[25]:
    Train MAE  Test MAE  Train MSE  Test MSE  Fit_
    time \
DecisionTreeRegressor    1.15315    3.05854    2.792049    21.477299    00:00:00

    Test time on train df  Test time on test df
DecisionTreeRegressor          00:00:00          00:00:00

```

```

[26]: model = RandomForestRegressor(max_depth=7, n_jobs=-1, random_state=42)

```

```

start = time.time()
model.fit(X_train, y_train)
end = time.time()
fitTime = exec_time(start, end)

start = time.time()
y_pred = model.predict(X_test)
end = time.time()
testTime = exec_time(start, end)

start = time.time()
y_train_pred = model.predict(X_train)
end = time.time()
trainTime = exec_time(start, end)

testMAE = mean_absolute_error(y_test, y_pred)
trainMAE = mean_absolute_error(y_train, y_train_pred)
testMSE = mean_squared_error(y_test, y_pred)
trainMSE = mean_squared_error(y_train, y_train_pred)
print("Test MAE = %.4f" % testMAE)
print("Train MAE = %.4f" % trainMAE)
print("Test MAE = %.4f" % testMSE)
print("Train MAE = %.4f" % trainMSE)

```

```

Test MAE = 2.4400
Train MAE = 1.3592
Test MAE = 12.6056
Train MAE = 3.1706

```

```
[27]: RandomForestRegressorMAE = pd.DataFrame({
    "Train MAE" : [trainMAE],
    "Test MAE" : [testMAE],
    "Train MSE" : [trainMSE],
    "Test MSE" : [testMSE],
    "Fit time" : [fitTime],
    "Test time on train df" : [trainTime],
    "Test time on test df" : [testTime],
}, index=["RandomForestRegressor"])
RandomForestRegressorMAE
```

```
[27]:
```

	Train MAE	Test MAE	Train MSE	Test MSE	Fit
RandomForestRegressor	1.359185	2.439958	3.170551	12.605613	00:00:00

	Test time on train df	Test time on test df
RandomForestRegressor	00:00:00	00:00:00

### 3.14 O

```
[28]: AllMAE = pd.concat([DecisionTreeRegressorMAE, RandomForestRegressorMAE])
AllMAE.sort_values(by=["Test MAE"])
```

```
[28]:
```

	Train MAE	Test MAE	Train MSE	Test MSE	Fit
RandomForestRegressor	1.359185	2.439958	3.170551	12.605613	00:00:00
DecisionTreeRegressor	1.153150	3.058540	2.792049	21.477299	00:00:00

	Test time on train df	Test time on test df
RandomForestRegressor	00:00:00	00:00:00
DecisionTreeRegressor	00:00:00	00:00:00

### 3.15 Выводы

Использовал MAE и MSE из-за простоты и популярности данных метрик. Полученные результаты говорят о том, что случайный лес справляется с переобучением лучше, чем дерево решений (ожидаемо). Однако обе модели плохо обучены из-за малого набора данных с большим количеством пропусков. Также не получилось исследовать скорость обучения и предсказания на малом наборе данных.