ТМО ЛР5 ИУ5-63Б Горкунов Николай

5 июня 2024 г.

1 *ТМО ЛР5 ИУ5-63Б Горкунов Николай*

2 Ансамбли моделей машинного обучения. Часть 1.

- Выберите набор данных (датасет) для решения задачи классификации или регресии.
- В случае необходимости проведите удаление или заполнение пропусков и кодирование категориальных признаков.
- С использованием метода train_test_split разделите выборку на обучающую и тестовую.
- Обучите следующие ансамблевые модели:
  - две модели группы бэггинга (бэггинг или случайный лес или сверхслучайные деревья);
  - AdaBoost;
  - градиентный бустинг.
- Оцените качество моделей с помощью одной из подходящих для задачи метрик. Сравните качество полученных моделей.иде.делей.

3 Набор данных: Boston housing dataset

[1]:
```
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
import numpy as np
from io import StringIO
from PIL import Image
from IPython.display import display
import graphviz
import pydotplus
from sklearn.tree import DecisionTreeRegressor, export_graphviz
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import BaggingRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.pipeline import make_pipeline
```

```
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/boston-housing-dataset
```

3.1 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
```

3.2 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
```

3.3 Проверяю значения категориальных признаков

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

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

3.4 Проверяю пропуски

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

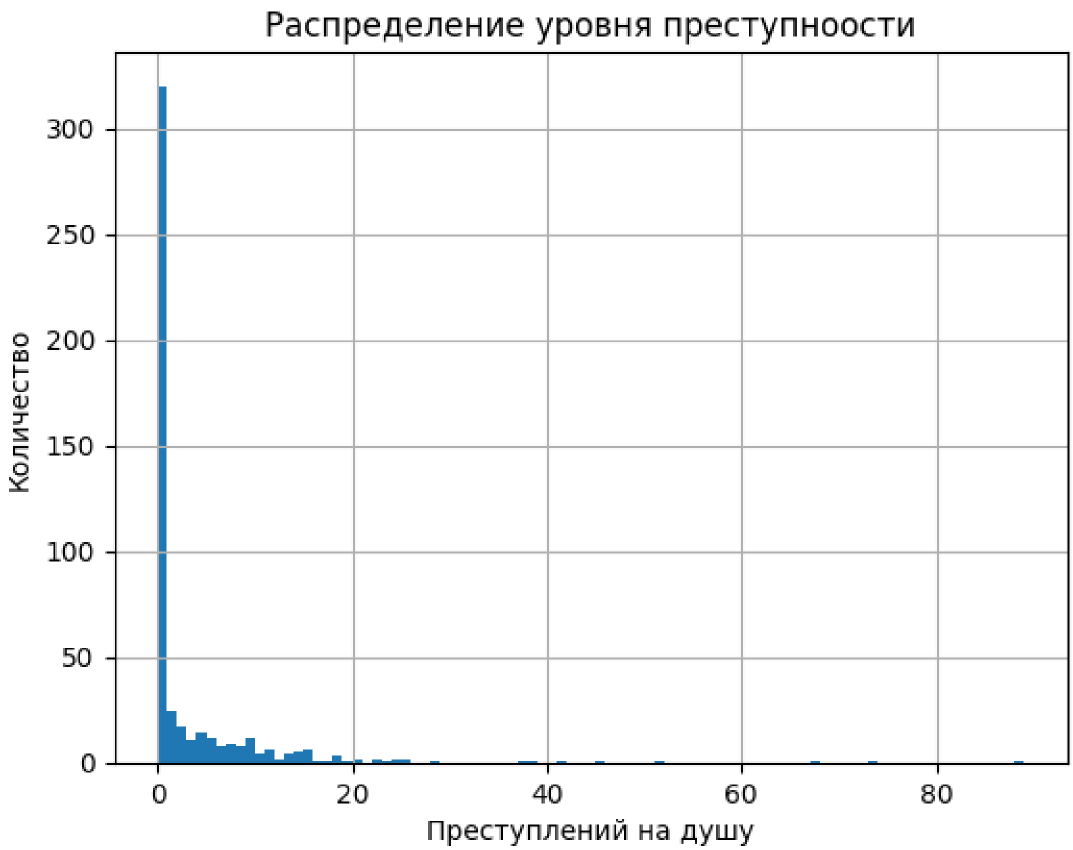
[6]:
```
CRIM       20
ZN         20
INDUS      20
CHAS       20
NOX         0
RM          0
AGE        20
DIS         0
RAD         0
TAX         0
PTRATIO     0
B           0
LSTAT      20
MEDV        0
dtype: int64
```

3.5 Заполняю пропуски в численном признаке "CRIM" в соответствии с описанием "CRIM - per capita crime rate by town"

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

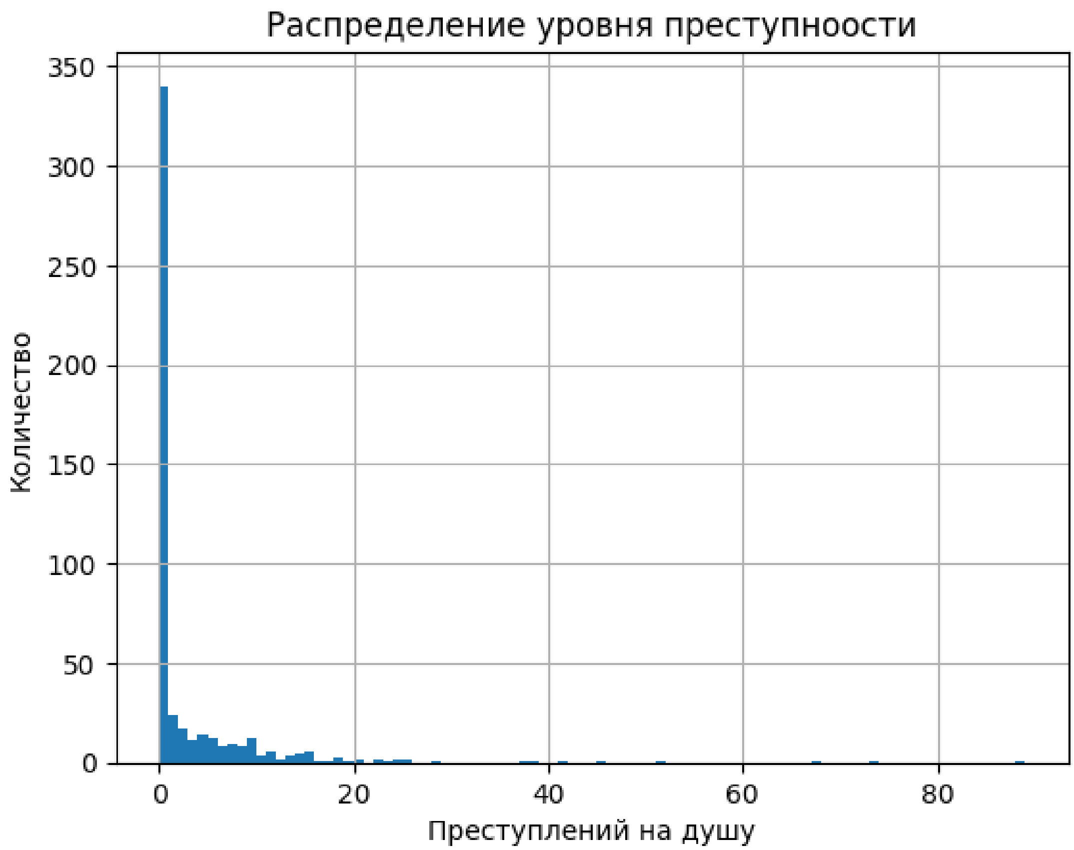
[7]:
```
Empty DataFrame
Columns: [CRIM, ZN, INDUS, CHAS, NOX, RM, AGE, DIS, RAD, TAX, PTRATIO, B, LSTAT,
MEDV]
Index: []
```

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



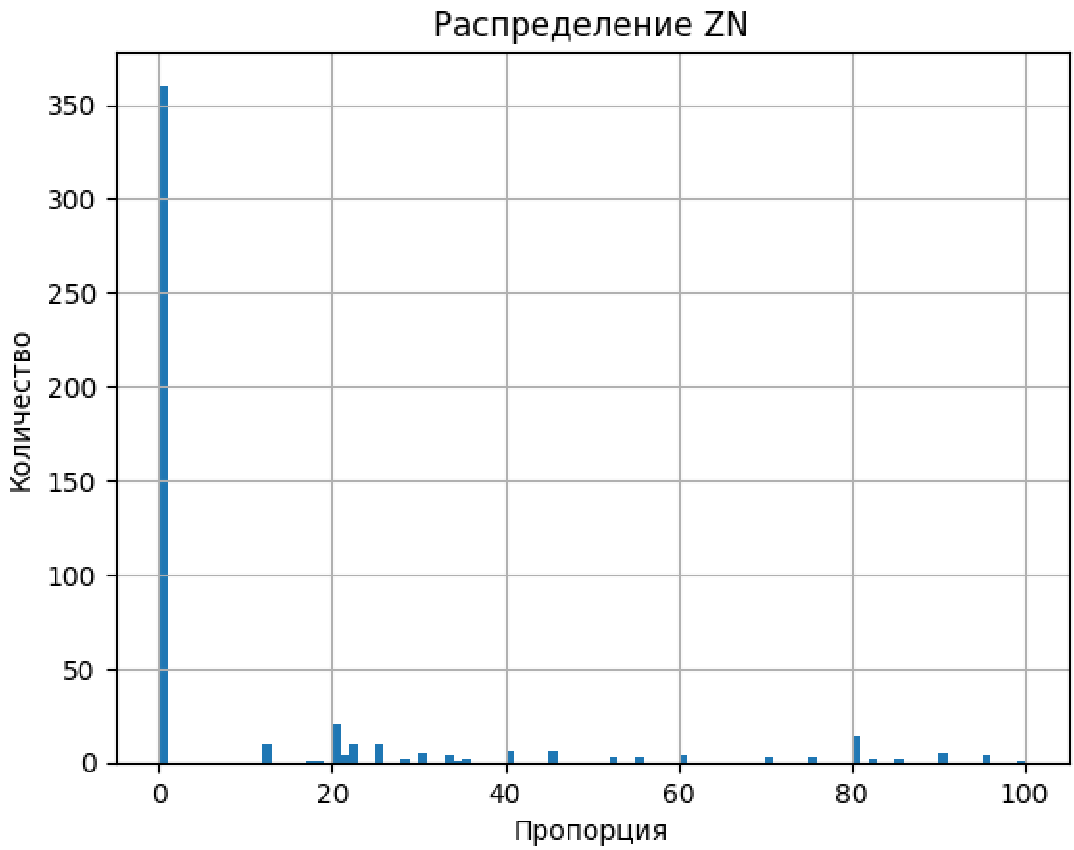
[9]:
```
df = df.fillna(value={"CRIM": 0})

df.CRIM.hist(bins=range(90))
plt.title('Распределение уровня преступноости')
plt.xlabel('Преступлений на душу')
plt.ylabel('Количество')
plt.show()
```



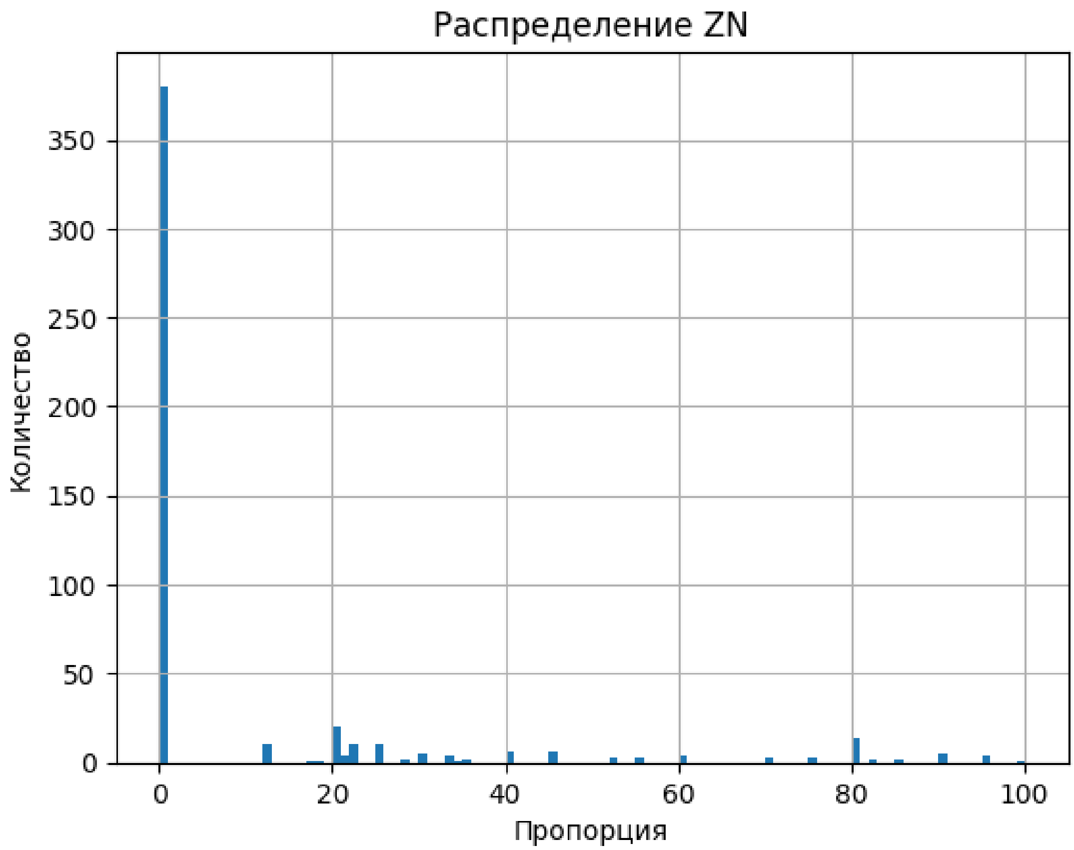
3.6 Заполняю пропуски в численном признаке "ZN" в соответствии с описанием "ZN - proportion of residential land zoned for lots over 25,000 sq.ft."

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



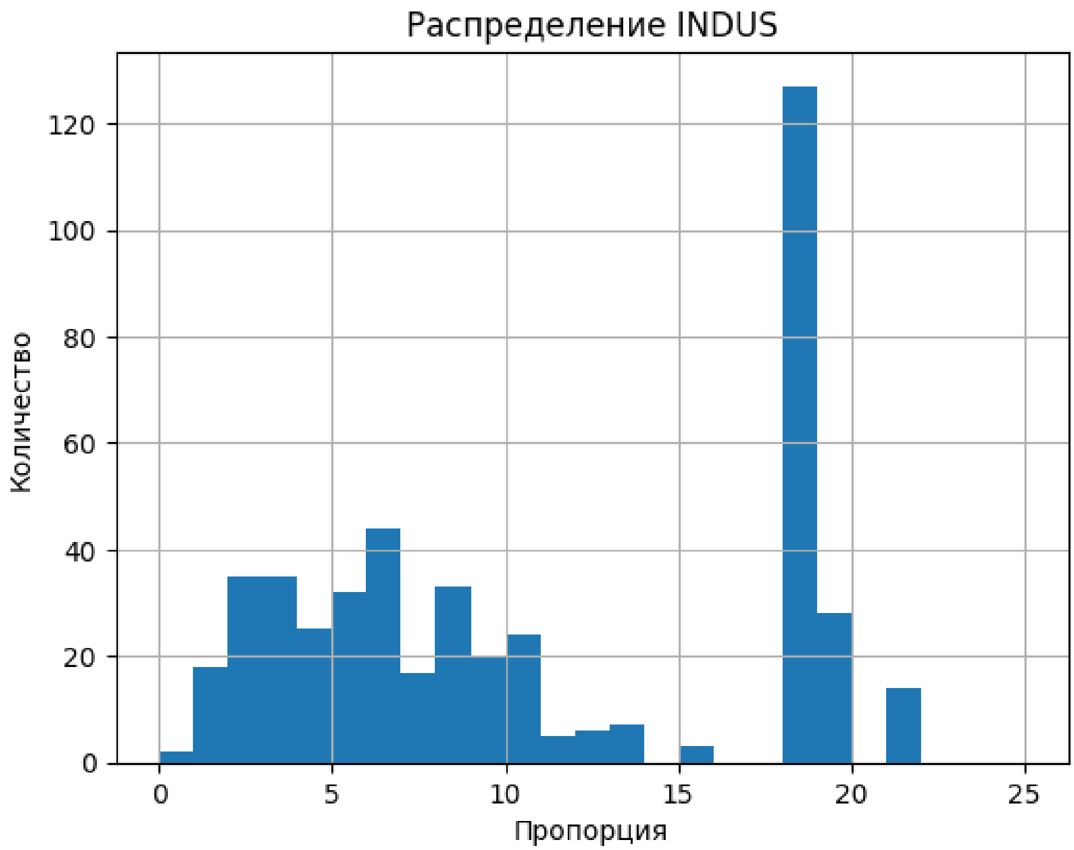
[11]:
```
df = df.fillna(value={"ZN": 0})

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



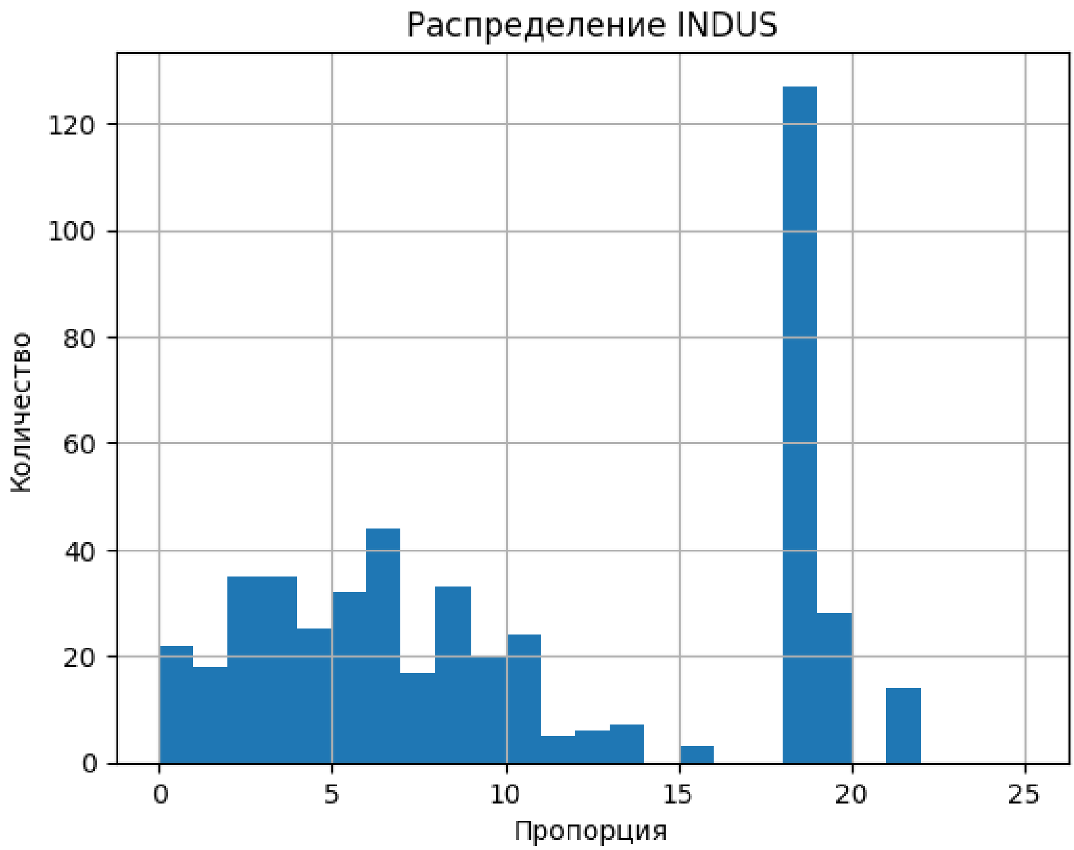
3.7 Заполняю пропуски в численном признаке "INDUS" в соответствии с описанием "INDUS - proportion of non-retail business acres per town."

[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()
```

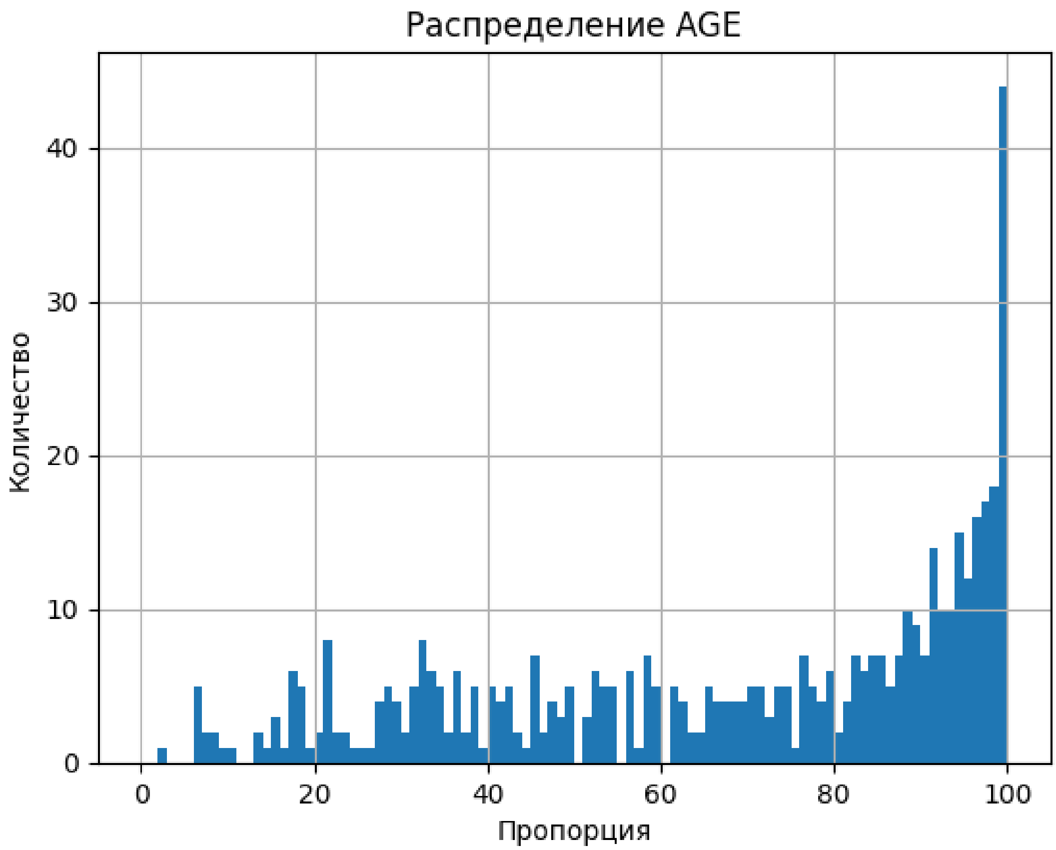


3.8 Не удаляю пропуски в категориальном признаке “CHAS” в соответствии с описанием “CHAS - Charles River dummy variable (1 if tract bounds river; 0 otherwise)”

[14]:
```
df = df.fillna(value={"CHAS": 2})
```

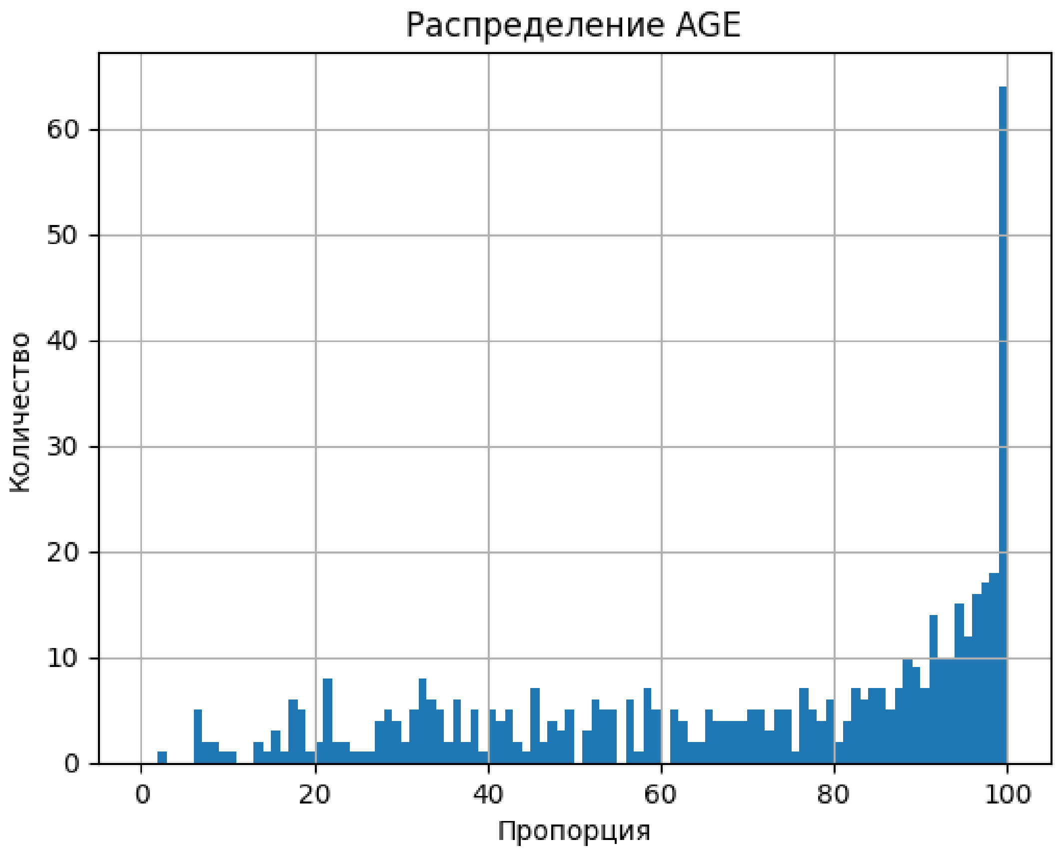
3.9 Заполняю пропуски в численном признаке “AGE” в соответствии с описанием “AGE - proportion of owner-occupied units built prior to 1940”

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



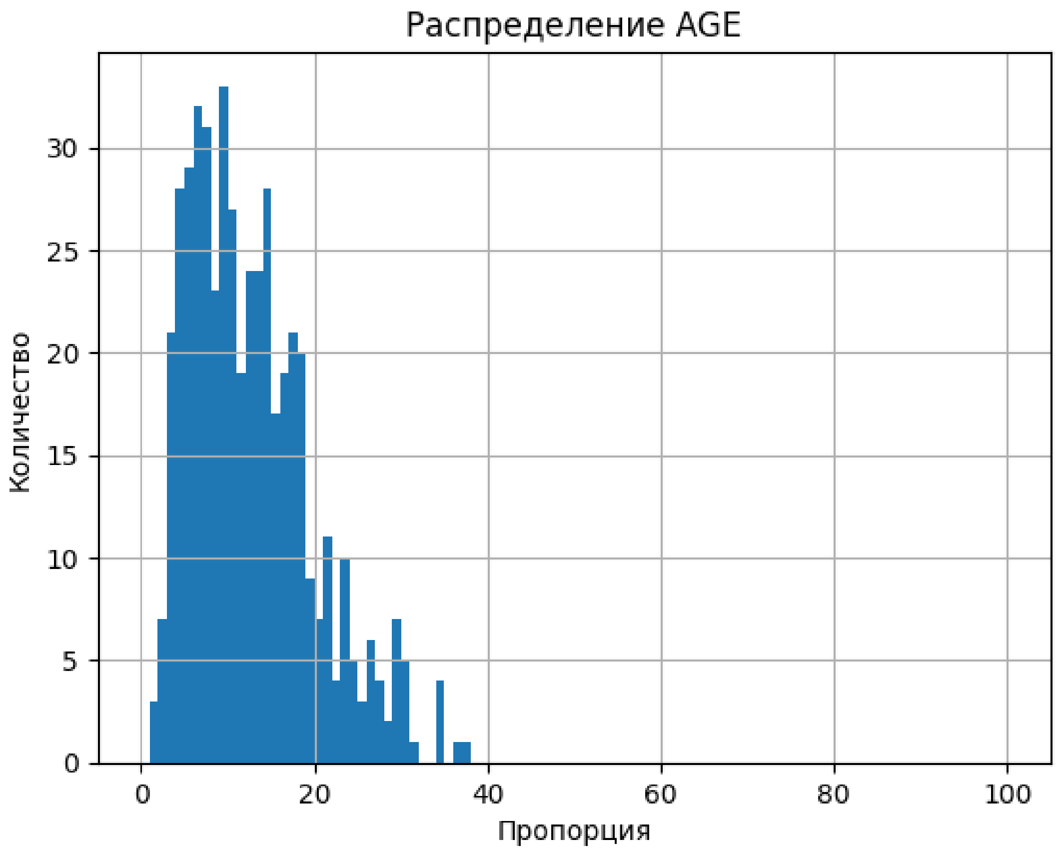
[16]:
```
df = df.fillna(value={"AGE": 100})

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



3.10 Заполняю пропуски в численном признаке "LSTAT" в соответствии с описанием "LSTAT - % lower status of the population"

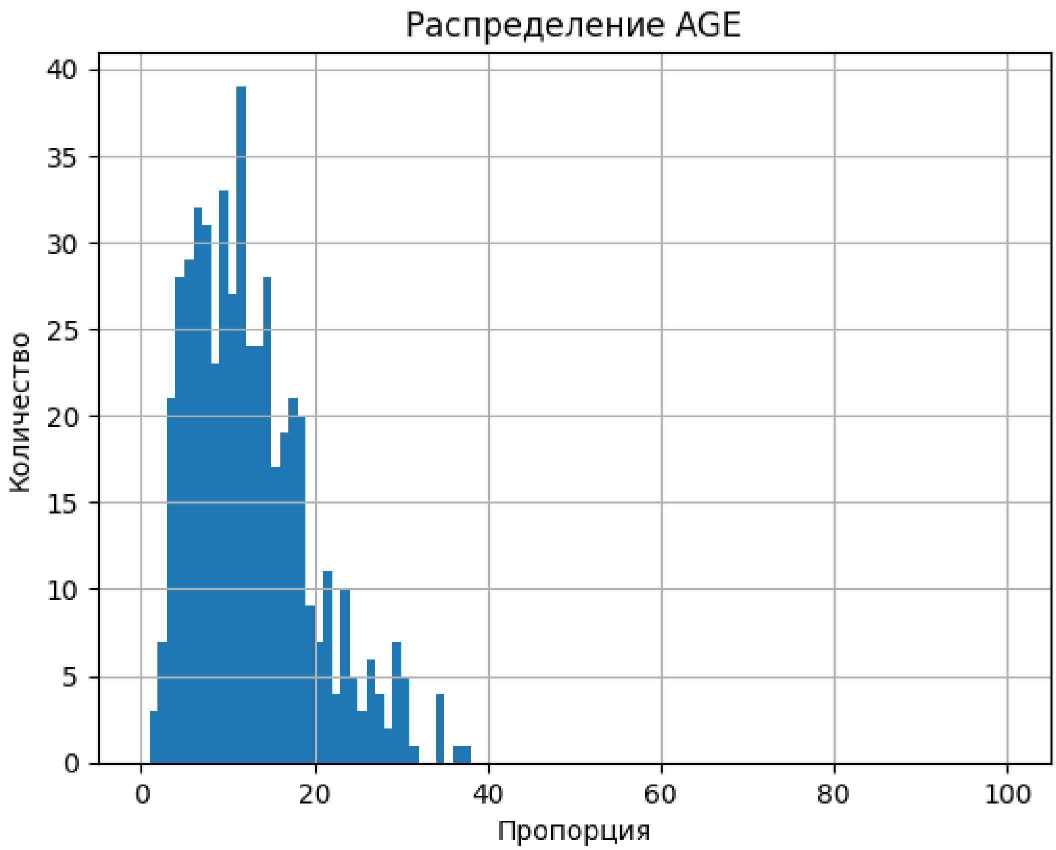
[17]:
```
df.LSTAT.hist(bins=range(101))
plt.title('Распределение AGE')
plt.xlabel('Пропорция')
plt.ylabel('Количество')
plt.show()
```



[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()
```

```
11.43
```



[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
```

3.11 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  2.0
0  396.90   4.98  24.0    1    0    0
1  396.90   9.14  21.6    1    0    0
2  392.83   4.03  34.7    1    0    0
3  394.63   2.94  33.4    1    0    0
4  396.90  11.00  36.2    1    0    0
```

3.12 Провожу разделение на тестовую и обучающую выборки, обучаю и тестирую KNN для предсказания признака MEDV (регрессия), оцениваю с 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)
```

3.13 RandomForestRegressor

[24]:
```
model = RandomForestRegressor(oob_score=True, 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 MSE = %.4f" % testMSE)
print("Train MSE = %.4f" % trainMSE)
```

```
Test MAE = 2.1483
Train MAE = 0.9802
Test MSE = 9.7985
Train MSE = 2.5207
```

[25]:

```
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
```

[25]:

```
                       Train MAE  Test MAE  Train MSE  Test MSE  Fit time  \
RandomForestRegressor   0.980153  2.148323   2.520687  9.798453  00:00:00

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

3.14 ExtraTreesRegressor

[26]:
```
model = ExtraTreesRegressor(bootstrap=True, oob_score=True, 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 MSE = %.4f" % testMSE)
print("Train MSE = %.4f" % trainMSE)
```

```
Test MAE = 2.0043
Train MAE = 0.9405
Test MSE = 11.5535
Train MSE = 2.0609
```

[27]:
```
ExtraTreesRegressorMAE = 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=["ExtraTreesRegressor"])
ExtraTreesRegressorMAE
```

[27]:
```
                     Train MAE  Test MAE  Train MSE   Test MSE  Fit time  \
ExtraTreesRegressor    0.94051  2.004269   2.060851  11.553487  00:00:00

                   Test time on train df Test time on test df
```

```
ExtraTreesRegressor                00:00:00                00:00:00
```

3.15 AdaBoostRegressor

[28]:
```
model = AdaBoostRegressor(n_estimators=100, 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 MSE = %.4f" % testMSE)
print("Train MSE = %.4f" % trainMSE)
```

```
Test MAE = 2.3307
Train MAE = 2.1889
Test MSE = 11.8726
Train MSE = 7.3984
```

[29]:
```
AdaBoostRegressorMAE = 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=["AdaBoostRegressor"])
AdaBoostRegressorMAE
```

[29]:

```
                   Train MAE  Test MAE  Train MSE   Test MSE  Fit time  \
AdaBoostRegressor   2.188942  2.330673   7.398446  11.872565  00:00:00

                  Test time on train df Test time on test df
AdaBoostRegressor              00:00:00             00:00:00
```

3.16 GradientBoostingRegressor

[30]:

```
model = GradientBoostingRegressor(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 MSE = %.4f" % testMSE)
print("Train MSE = %.4f" % trainMSE)
```

```
Test MAE = 2.0407
Train MAE = 1.0978
Test MSE = 8.4135
Train MSE = 1.9931
```

[31]:

```
GradientBoostingRegressorMAE = 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=["GradientBoostingRegressor"])
```

```
GradientBoostingRegressorMAE
```

[31]:

```
                           Train MAE  Test MAE  Train MSE  Test MSE  Fit time  \
GradientBoostingRegressor   1.097762   2.04074    1.99313  8.413547  00:00:00

                          Test time on train df Test time on test df
GradientBoostingRegressor              00:00:00             00:00:00
```

3.17 Провожу сравнение

[32]:

```
AllMAE = pd.concat([RandomForestRegressorMAE, ExtraTreesRegressorMAE,␣
 ↪AdaBoostRegressorMAE, GradientBoostingRegressorMAE])
AllMAE.sort_values(by=["Test MSE"])
```

[32]:

```
                           Train MAE  Test MAE  Train MSE   Test MSE  \
GradientBoostingRegressor   1.097762  2.040740   1.993130   8.413547
RandomForestRegressor       0.980153  2.148323   2.520687   9.798453
ExtraTreesRegressor         0.940510  2.004269   2.060851  11.553487
AdaBoostRegressor           2.188942  2.330673   7.398446  11.872565

                           Fit time Test time on train df Test time on test df
GradientBoostingRegressor  00:00:00              00:00:00             00:00:00
RandomForestRegressor      00:00:00              00:00:00             00:00:00
ExtraTreesRegressor        00:00:00              00:00:00             00:00:00
AdaBoostRegressor          00:00:00              00:00:00             00:00:00
```