ТМО_ЛР6_ИУ5_63Б_Горкунов_Николай

5 июня 2024 г.

1 *ТМО ЛР6 ИУ5-63Б Горкунов Николай*

2 Ансамбли моделей машинного обучения. Часть 2.

- 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:
  - одну из моделей группы стекинга.
  - модель многослойного персептрона. По желанию, вместо библиотеки scikit-learn возможно использование библиотек TensorFlow, PyTorch или других аналогичных библиотек.
  - двумя методами на выбор из семейства МГУА (один из линейных методов COMBI / MULTI + один из нелинейных методов MIA / RIA) с использованием библиотеки gmdh. В настоящее время библиотека МГУА не позволяет решать задачу классификации !!!
- Оцените качество моделей с помощью одной из подходящих для задачи метрик. Сравните качество полученных моделей.е.делей.

3 Набор данных: Boston housing dataset

[36]:
```
# %pip install gmdh
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to `transformed_cell`
argument and any exception that happen during thetransform in
`preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

[37]:
```
import warnings
warnings.filterwarnings("ignore")
import gmdh
import pandas as pd
import numpy as np
from sklearn.ensemble import ExtraTreesRegressor, StackingRegressor
```

```
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.linear_model import SGDRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.neural_network import MLPRegressor
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
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to `transformed_cell`
argument and any exception that happen during thetransform in
`preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

[38]:
```
#kaggle_api = KaggleApi()
#kaggle_api.authenticate()
#kaggle_api.dataset_download_files('altavish/boston-housing-dataset', unzip=True)
```

3.1 Смотрю, что в данных

[39]:
```
df = pd.read_csv('HousingData.csv')
print(df.shape)
df.head()
```

```
(506, 14)
```

[39]:
```
      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

[40]:
```
df.dtypes
```

[40]:
```
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

[41]:
```
df.CHAS.unique()
```

[41]:
```
array([ 0., nan,  1.])
```

3.4 Проверяю пропуски

[42]:
```
df.isna().sum()
```

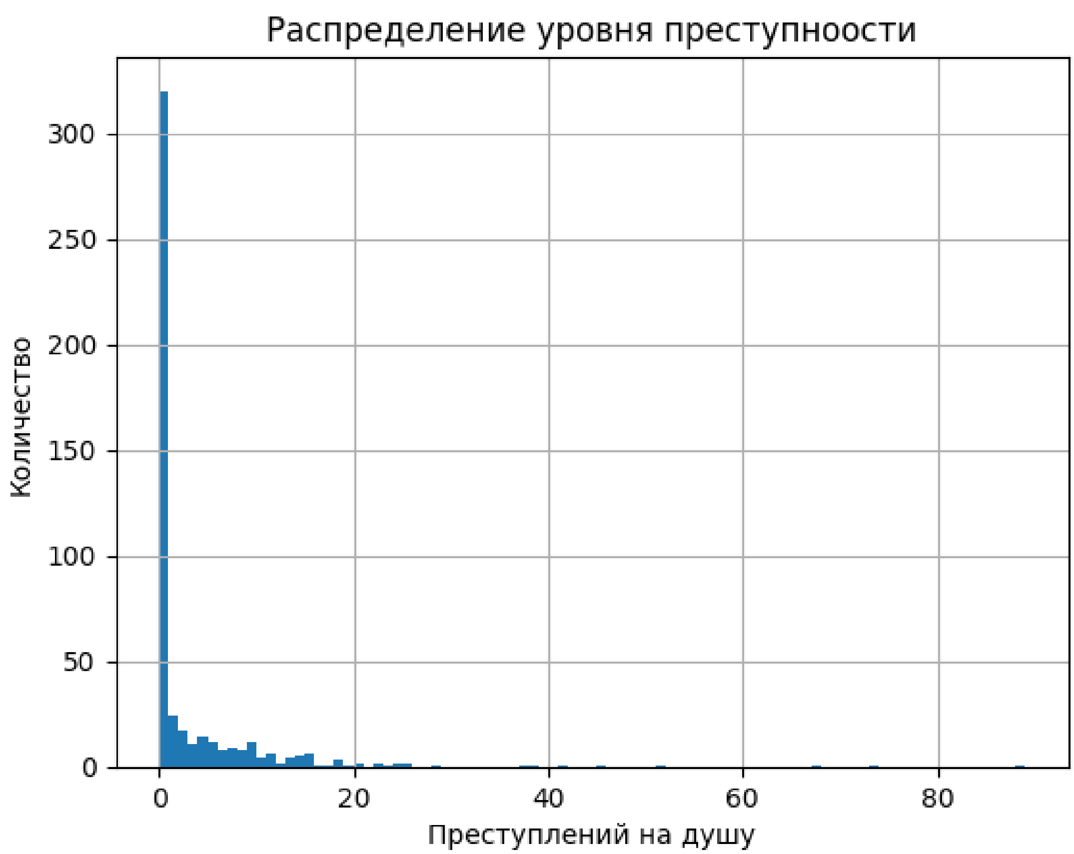
[42]:
```
CRIM       20
ZN         20
INDUS      20
CHAS       20
NOX         0
RM          0
AGE        20
DIS         0
RAD         0
TAX         0
PTRATIO     0
B           0
LSTAT      20
MEDV        0
dtype: int64
```

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

[43]:
```
df[df.CRIM == 0]
```

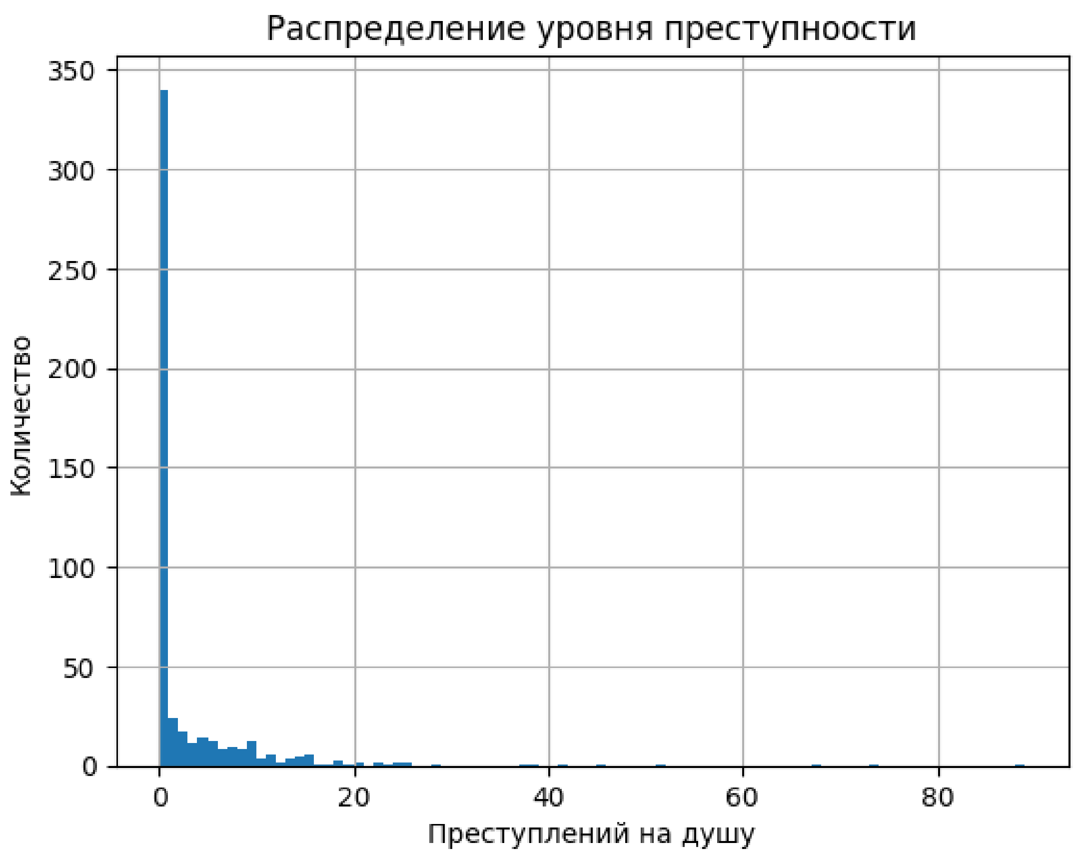
[43]:
```
Empty DataFrame
Columns: [CRIM, ZN, INDUS, CHAS, NOX, RM, AGE, DIS, RAD, TAX, PTRATIO, B, LSTAT,
MEDV]
Index: []
```

[44]:
```
df.CRIM.hist(bins=range(90))
plt.title('Распределение уровня преступноости')
plt.xlabel('Преступлений на душу')
plt.ylabel('Количество')
plt.show()
```



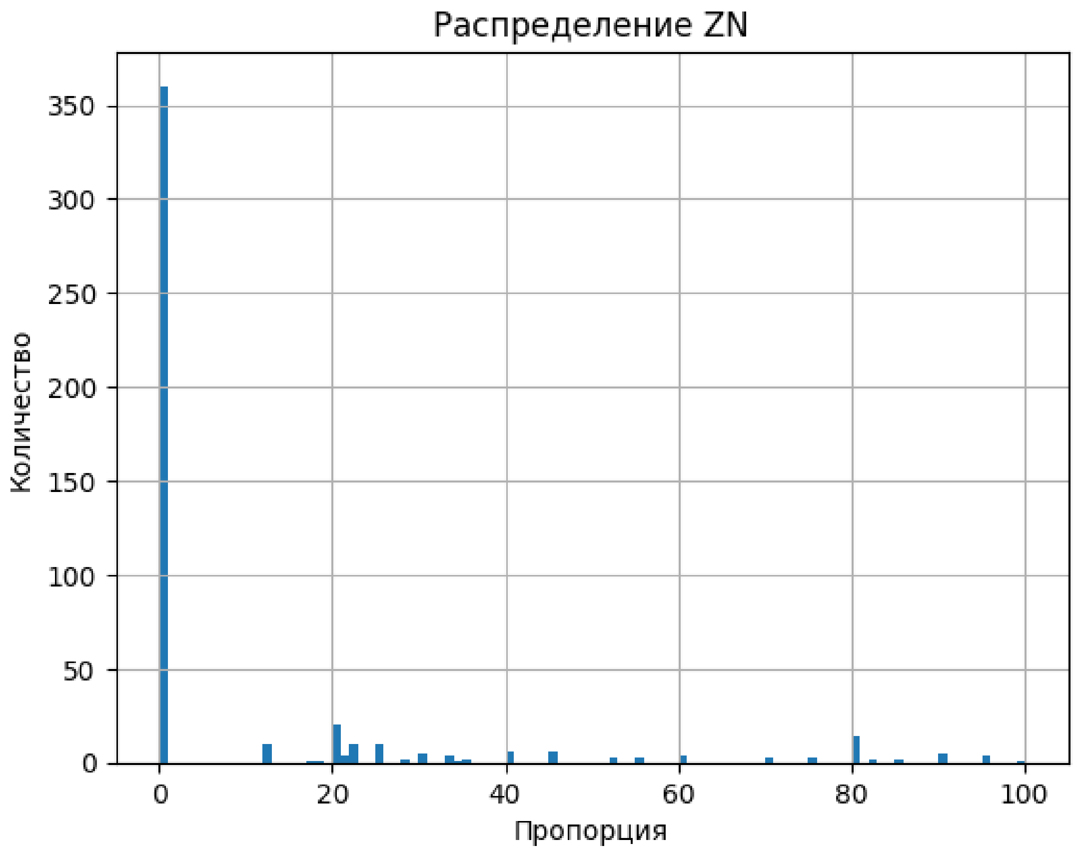
[45]:
```
df = df.fillna(value={"CRIM": 0})

df.CRIM.hist(bins=range(90))
plt.title('Распределение уровня преступноости')
```

```
plt.xlabel('Преступлений на душу')
plt.ylabel('Количество')
plt.show()
```

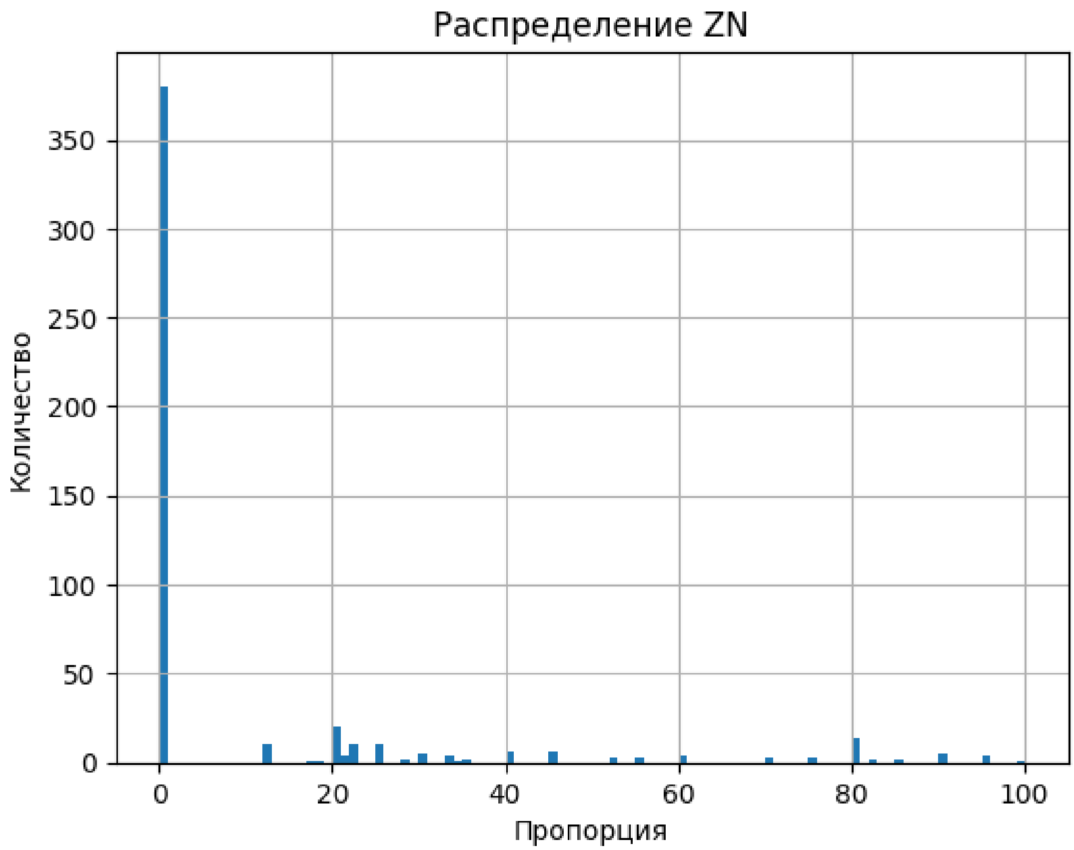


3.6 Заполняю пропуски в численном признаке “ZN” в соответствии с описанием “ZN - proportion of residential land zoned for lots over 25,000 sq.ft.”

[46]:
```
df.ZN.hist(bins=range(101))
plt.title('Распределение ZN')
plt.xlabel('Пропорция')
plt.ylabel('Количество')
plt.show()
```



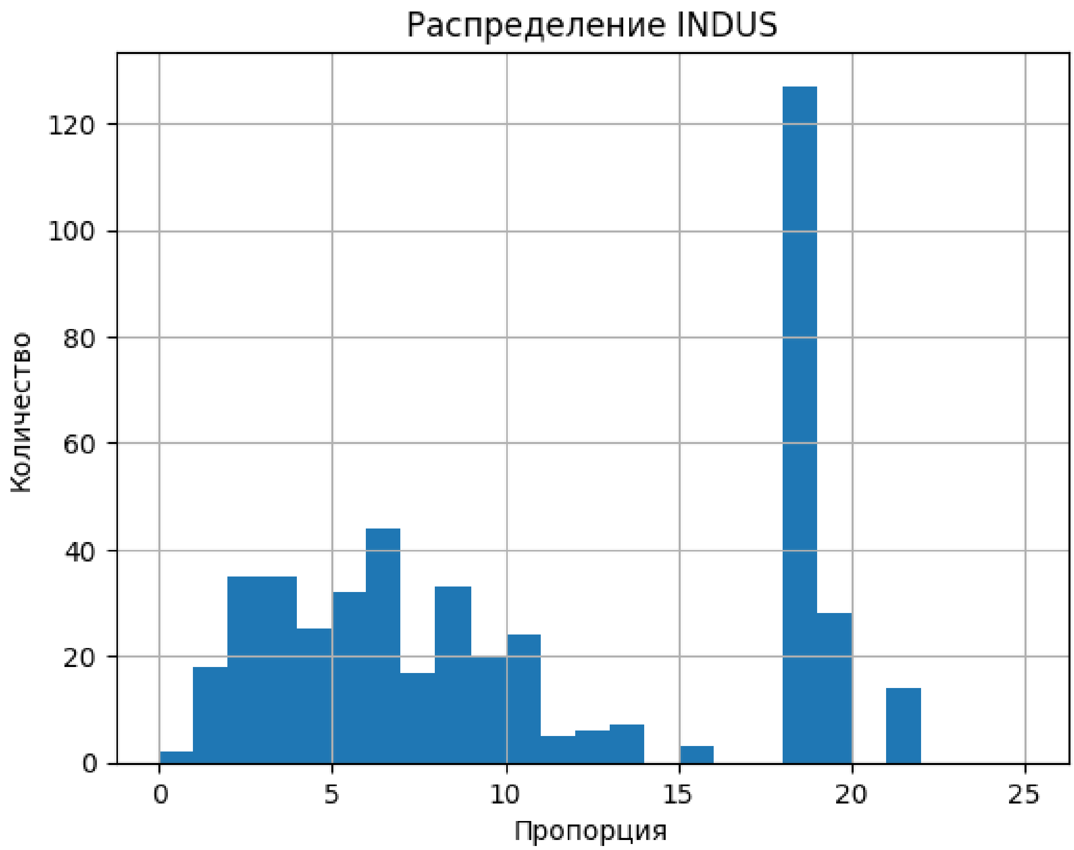
[47]:
```
df = df.fillna(value={"ZN": 0})

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



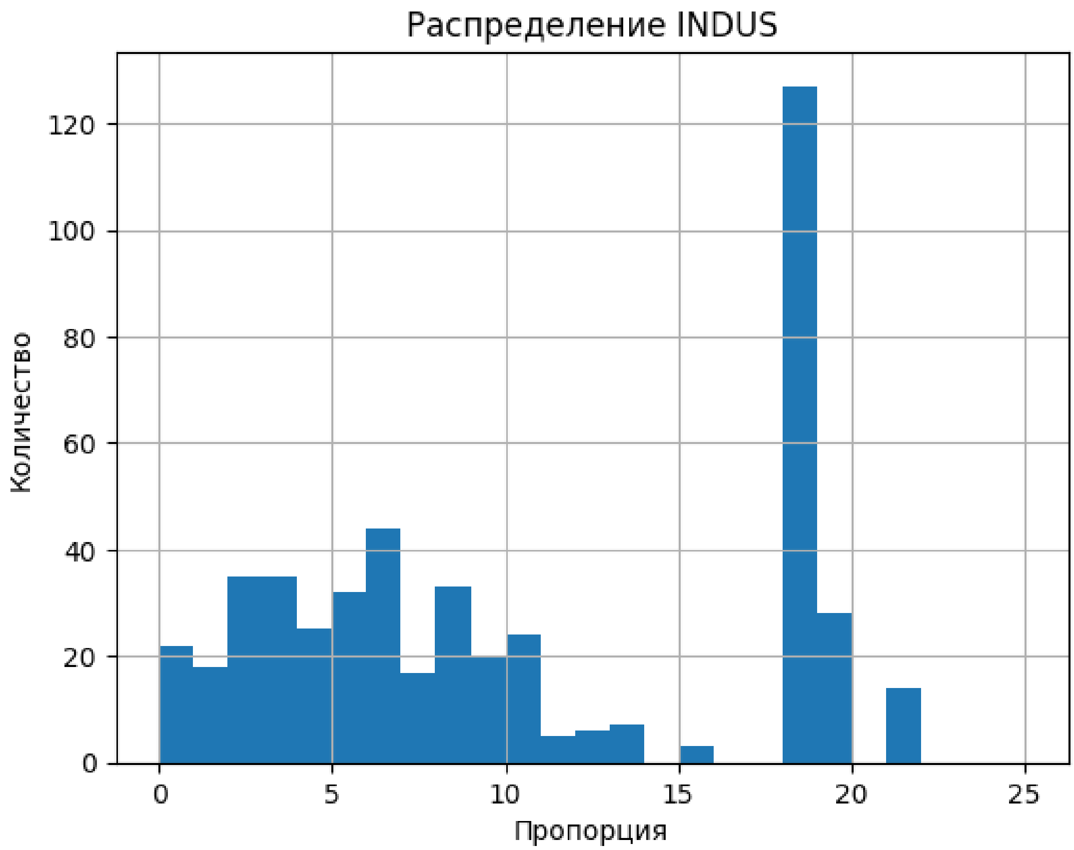
3.7 Заполняю пропуски в численном признаке "INDUS" в соответствии с описанием "INDUS - proportion of non-retail business acres per town."

[48]:

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



[49]:
```
df = df.fillna(value={"INDUS": 0})

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

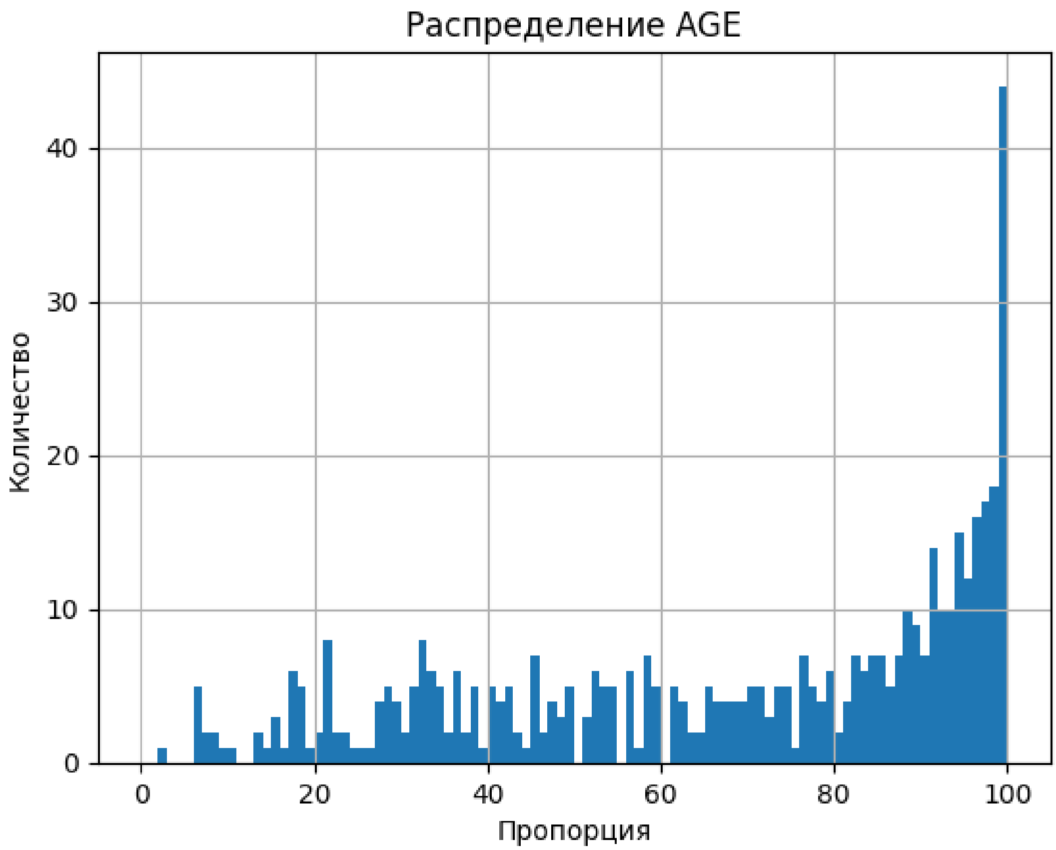


3.8 Не удаляю пропуски в категориальном признаке "CHAS" в соответствии с описанием "CHAS - Charles River dummy variable (1 if tract bounds river; 0 otherwise)"

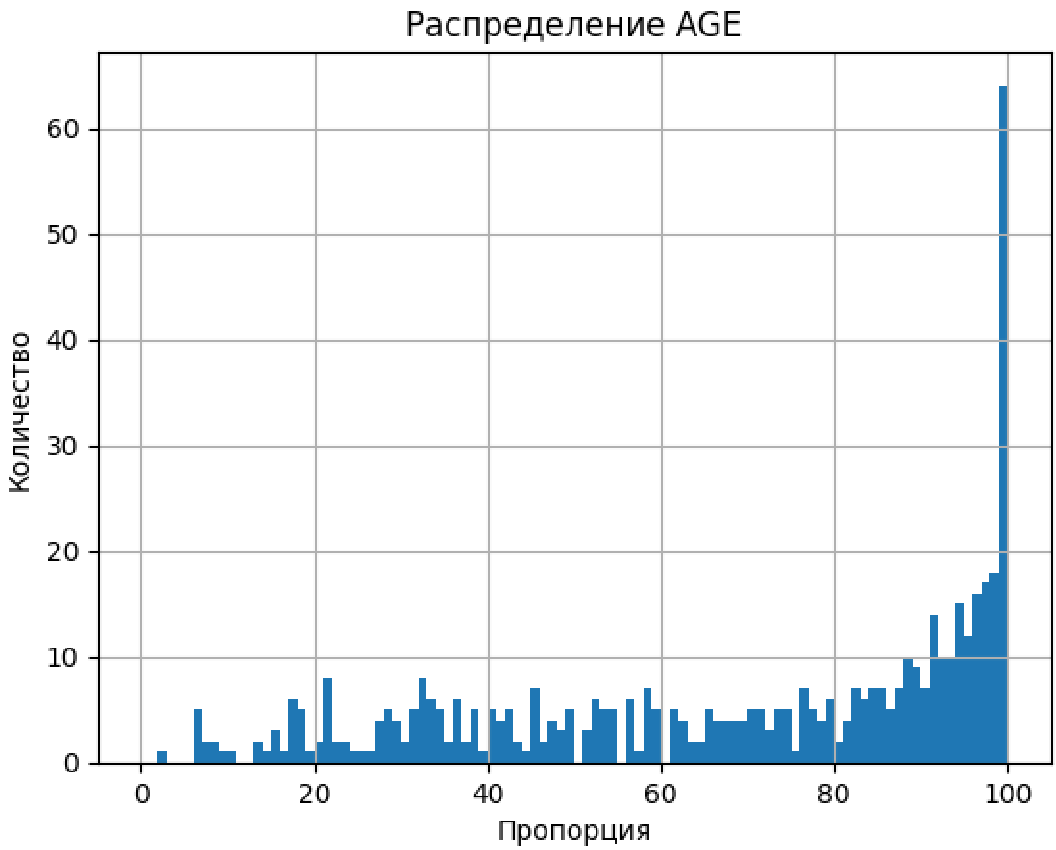
[50]:
```
df = df.fillna(value={"CHAS": 2})
```

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

[51]:
```
df.AGE.hist(bins=range(101))
plt.title('Распределение AGE')
plt.xlabel('Пропорция')
plt.ylabel('Количество')
plt.show()
```

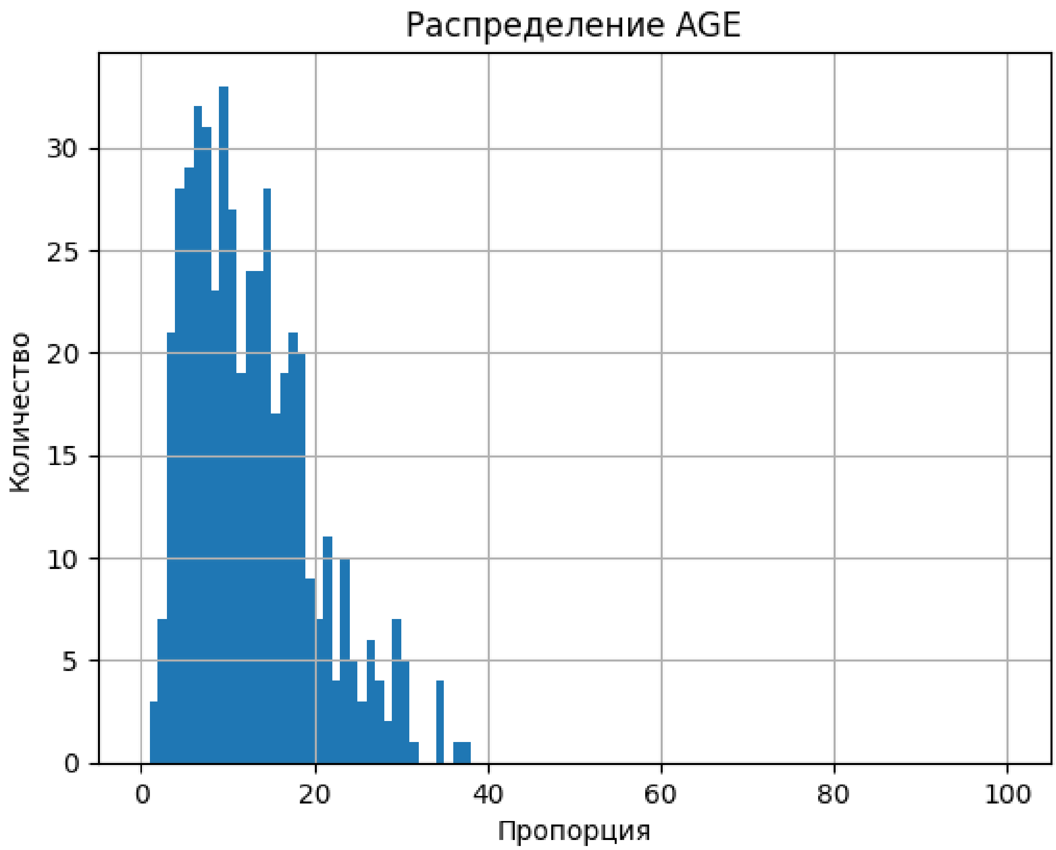


[52]:
```
df = df.fillna(value={"AGE": 100})

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



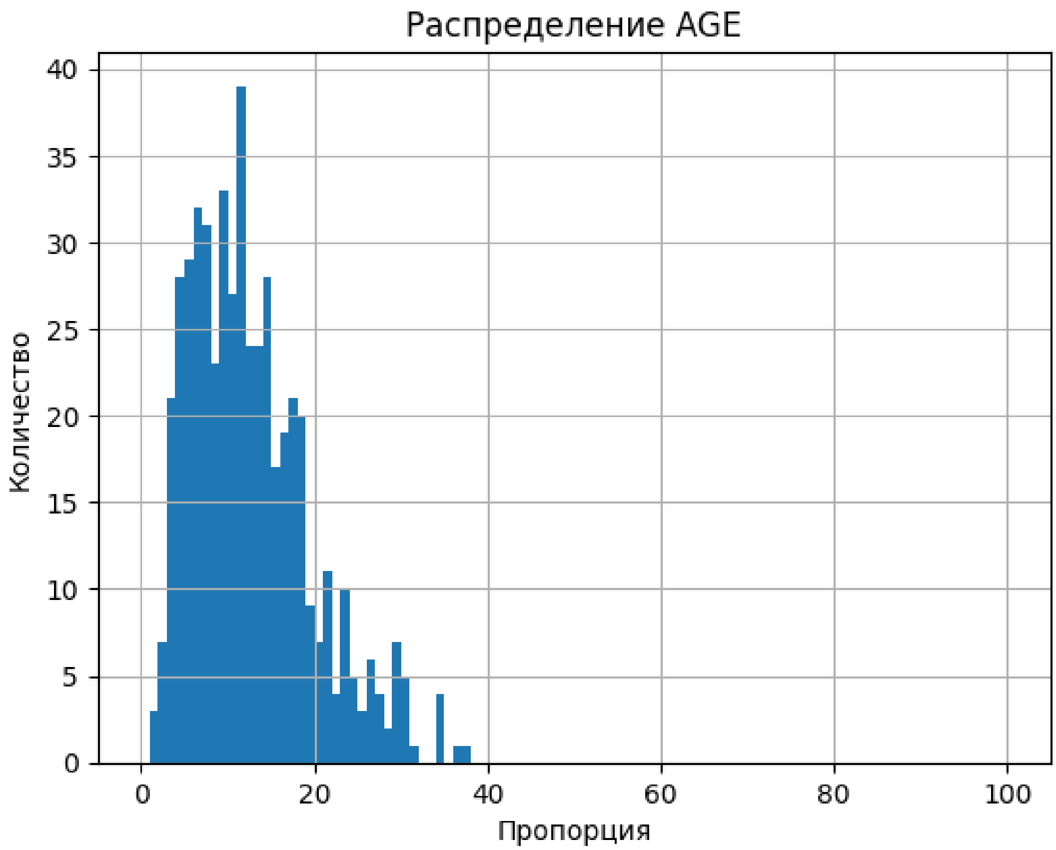
3.10 Заполняю пропуски в численном признаке "LSTAT" в соответствии с описанием "LSTAT - % lower status of the population"

[53]:
```
df.LSTAT.hist(bins=range(101))
plt.title('Распределение AGE')
plt.xlabel('Пропорция')
plt.ylabel('Количество')
plt.show()
```



[54]:
```
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



[55]:
```
df.isna().sum()
```

[55]:
```
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)

[56]:
```
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()
```

[56]:
```
      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

[57]:
```
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))
```

[58]:
```
y = df.MEDV.copy()
X = df.loc[:, df.columns != "MEDV"].copy()
```

[59]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,␣
 ↪random_state=42)
```

3.13 StackingRegressor

[60]:
```
gb = GradientBoostingRegressor(random_state=42)
sgd = make_pipeline(StandardScaler(), SGDRegressor(learning_rate="adaptive",␣
 ↪random_state=42))
knn = KNeighborsRegressor(n_jobs=-1, n_neighbors=7)
model = StackingRegressor(
```

```
    estimators=[('sgd', sgd), ('gb', gb), ('knn', knn)],
    final_estimator=ExtraTreesRegressor(n_jobs=-1, n_estimators=42,␣
↪max_depth=None, 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 = 2.3537
Train MAE = 1.8159
Test MSE = 12.5758
Train MSE = 7.5975
```

[61]:
```
StackingRegressorMAE = 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=["StackingRegressor"])
StackingRegressorMAE
```

[61]:
```
                     Train MAE  Test MAE  Train MSE   Test MSE  Fit time  \
StackingRegressor     1.815922  2.353693   7.597507  12.575775  00:00:01
```

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

3.14 MLPRegressor

[62]:
```python
model = MLPRegressor(max_iter=1234, 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 = 3.0604
Train MAE = 3.1309
Test MSE = 20.4412
Train MSE = 20.5303
```

[63]:
```python
MLPRegressorMAE = 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=["MLPRegressor"])
MLPRegressorMAE
```

[63]:
```
              Train MAE  Test MAE  Train MSE   Test MSE  Fit time  \
MLPRegressor   3.130907  3.060416  20.530314  20.441244  00:00:01

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

3.15 GmdhMia

[64]:
```
X_train.isna().sum()
```

[64]:
```
CRIM       0
ZN         0
INDUS      0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
0.0        0
1.0        0
2.0        0
dtype: int64
```

[65]:
```
y_train.isna().sum()
```

[65]:
```
0
```

[66]:
```
model = gmdh.Mia()

start = time.time()
model.fit(X_train, y_train, polynomial_type=gmdh.PolynomialType.LINEAR,
          criterion=gmdh.Criterion(criterion_type=gmdh.CriterionType.
 ↪SYM_REGULARITY),
          n_jobs=-1, k_best=20, limit=0.01)
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()
```

```python
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)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-66-c386c40fae1e> in <cell line: 4>()
      2
      3 start = time.time()
----> 4 model.fit(X_train, y_train, polynomial_type=gmdh.PolynomialType.LINEAR,
      5           criterion=gmdh.Criterion(criterion_type=gmdh.CriterionType.
↪SYM_REGULARITY),
      6           n_jobs=-1, k_best=20, limit=0.01)

/usr/local/lib/python3.10/dist-packages/gmdh/gmdh.py in fit(self, X, y, criterion,␣
↪k_best, polynomial_type, test_size, p_average, n_jobs, verbose, limit)
    850             Fitted model.
    851         """
--> 852         super().fit(X, y)
    853         self._model.fit(X, y, criterion._get_core(), k_best,
    854             _gmdh_core.PolynomialType(polynomial_type.value), test_size,

/usr/local/lib/python3.10/dist-packages/gmdh/gmdh.py in fit(self, X, y)
    392         """
    393
--> 394         if np.isnan(X).sum() > 0:
    395             raise ValueError('X array contains nan values')
    396         if np.isnan(y).sum() > 0:

/usr/local/lib/python3.10/dist-packages/pandas/core/generic.py in __nonzero__(self)
   1464     @final
   1465     def __nonzero__(self) -> NoReturn:
-> 1466         raise ValueError(
   1467             f"The truth value of a {type(self).__name__} is ambiguous. "
   1468             "Use a.empty, a.bool(), a.item(), a.any() or a.all()."
```

```
ValueError: The truth value of a Series is ambiguous. Use a.empty, a.bool(), a.
↪item(), a.any() or a.all().
```

[ ]:
```
GmdhMiaMAE = 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=["GmdhMia"])
GmdhMiaMAE
```

3.16 GmdhCombi

[ ]:
```
model = gmdh.Combi()

start = time.time()
model.fit(X_train, y_train, n_jobs=-1, test_size=0.24, limit=0, criterion=gmdh.
↪Criterion(gmdh.CriterionType.REGULARITY))
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)
```

[ ]:
```
GmdhCombiMAE = 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=["GmdhCombi"])
GmdhCombiMAE
```

3.17 Провожу сравнение

[67]:
```
#AllMAE = pd.concat([StackingRegressorMAE, MLPRegressorMAE, GmdhMiaMAE,␣
 ↪GmdhCombiMAE])
AllMAE = pd.concat([StackingRegressorMAE, MLPRegressorMAE])
AllMAE.sort_values(by=["Test MSE"])
```

[67]:
```
                     Train MAE  Test MAE  Train MSE   Test MSE  Fit time  \
StackingRegressor     1.815922  2.353693   7.597507  12.575775  00:00:01
MLPRegressor          3.130907  3.060416  20.530314  20.441244  00:00:01

                    Test time on train df Test time on test df
StackingRegressor                00:00:00             00:00:00
MLPRegressor                     00:00:00             00:00:00
```