**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Технологии машинного обучения»

Отчет по лабораторной работе №5

«Ансамбли моделей машинного обучения. Часть 1»

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-62Б |  | преподаватель каф. ИУ5 |
| Щепетов Дмитрий |  | Гапанюк Ю.Е. |
| Подпись и дата: |  | Подпись и дата: |

2024 г.

**Описание задания**

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

**Текст программы**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

from sklearn.model\_selection import train\_test\_split

import matplotlib.ticker as ticker

from sklearn.neighbors import KNeighborsRegressor, KNeighborsClassifier

import math

from enum import Enum

import os

# from plotly.express import line

#import data

#import plotly.express as px

# %matplotlib inline

sns.set(style="ticks")

plt.style.use('ggplot')

pd.set\_option('display.max\_columns',24)

pd.set\_option('display.max\_colwidth',None)

pd.set\_option('display.float\_format', lambda x: '%.4f' % x)

encoding = 'windows-1250'

data=pd.read\_csv(r'C:\Users\Dima\Desktop\Homework\3 курс\ТМО\ЛР5\melb\_data.csv', encoding = encoding)

data.shape

data.info()

type\_mapping = {

"h": "house",

"u": "unit",

"t": "townhouse"

}

# Map the values in the "Type" column

data["Type"] = data["Type"].map(type\_mapping)

method = { "S": "sold",

"SP" : "sold prior",

"PI" : "passed in",

"VB" : "vendor bid"}

data['Method'] = data['Method'].map(method)

null\_columns = data.columns[data.isnull().any()]

data.isnull().sum()

# Удаление строк с пропущенными значениями в столбцах "Method" и "Car"

data.dropna(subset=["Method", "Car"], inplace=True)

# data.dropna(subset=["Car"], inplace=True)

# Заполнение пропущенных значений в столбцах "BuildingArea", "YearBuilt" и "CouncilArea"

# Для числовых столбцов можно использовать медиану

median\_BuildingArea = data["BuildingArea"].median()

median\_YearBuilt = data["YearBuilt"].median()

# Для категориального столбца можно использовать наиболее часто встречающееся значение

mode\_CouncilArea = data["CouncilArea"].mode()[0]

data["BuildingArea"].fillna(median\_BuildingArea, inplace=True)

data["YearBuilt"].fillna(median\_YearBuilt, inplace=True)

data["CouncilArea"].fillna(mode\_CouncilArea, inplace=True)

data.hist(bins = 10,figsize=(20,12),grid=True)

plt.figure(figsize=(20,6))

plt.subplot(1,2,1)

data.Price[data.Type == 'house'].plot(kind='kde')

data.Price[data.Type == 'unit'].plot(kind='kde')

data.Price[data.Type == 'townhouse'].plot(kind='kde')

plt.xlabel('Price')

plt.title('Распределение цен на различные типы домов')

plt.legend(('house','unit','townhouse'),loc='best')

plt.subplot(1,2,2)

data.Propertycount[data.Type == 'house'].plot(kind='kde')

data.Propertycount[data.Type == 'unit'].plot(kind='kde')

data.Propertycount[data.Type == 'townhouse'].plot(kind='kde')

plt.xlabel('Количество объектов недвижимости')

plt.title('Распределение количества объектов недвижимости по различным типам домов')

plt.legend(('house','unit','townhouse'),loc='best')

from sklearn.model\_selection import train\_test\_split

# Определение признаков и целевой переменной

X = data.drop(columns=["Price"]) # Признаки (все столбцы кроме "Price")

y = data["Price"] # Целевая переменная (столбец "Price")

# Разделение выборки на обучающую и тестовую

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Вывод размеров обучающей и тестовой выборок

print("Размер обучающей выборки:", X\_train.shape[0])

print("Размер тестовой выборки:", X\_test.shape[0])

# Определение признаков (X) и целевой переменной (y)

X = data.drop(columns=["Price"]) # Все признаки, кроме Price

y = data["Price"] # Целевая переменная

from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline

from sklearn.ensemble import BaggingRegressor

from sklearn.tree import DecisionTreeRegressor

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import r2\_score

# Drop non-numeric columns

numeric\_columns = data.select\_dtypes(include=['float64', 'int64']).columns

X = data[numeric\_columns]

y = data["Price"]

# Split data into train and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Define and fit the model

br1 = BaggingRegressor(estimator=DecisionTreeRegressor(), n\_estimators=5, random\_state=10)

br1.fit(X\_train, y\_train)

# Evaluate the model

train\_score = br1.score(X\_train, y\_train)

test\_score = br1.score(X\_test, y\_test)

print("Train R^2 Score:", train\_score)

print("Test R^2 Score:", test\_score)

# Получение информации о выбранных образцах для каждого дерева

for i, samples in enumerate(br1.estimators\_samples\_):

print(f"Дерево {i+1}: {len(samples)} образцов")

tree\_samples = [samples for samples in br1.estimators\_samples\_]

print(tree\_samples)

# Сконвертируем эти данные в двоичную матрицу,

# 1 соответствует элементам, попавшим в обучающую выборку

bin\_array = np.zeros((5, X.shape[0]))

for i in range(5):

for j in br1.estimators\_samples\_[i]:

bin\_array[i][j] = 1

bin\_array

# И визуализируем (синим цветом показаны данные, которые попали в обучающую выборку)

fig, ax = plt.subplots(figsize=(12,2))

ax.pcolor(bin\_array, cmap='YlGnBu')

plt.show()

# Оценим Out-of-bag error, теоретическое значение 37%

for i in range(5):

cur\_data = bin\_array[i]

len\_cur\_data = len(cur\_data)

sum\_cur\_data = sum(cur\_data)

(len(bin\_array[0]) - sum(bin\_array[0])) / len(bin\_array[0])

oob\_i = (len\_cur\_data - sum\_cur\_data) / len\_cur\_data

print('Для модели № {} размер OOB составляет {}%'.format(i+1, round(oob\_i, 4)\*100.0))

from sklearn.ensemble import RandomForestRegressor, ExtraTreesRegressor

# Случайный лес

rf\_model = RandomForestRegressor(n\_estimators=100, random\_state=42)

rf\_model.fit(X\_train, y\_train)

def plot\_feature\_importances(model, feature\_names):

feature\_importances = model.feature\_importances\_

sorted\_indices = feature\_importances.argsort()

plt.figure(figsize=(10, 6))

plt.barh(range(len(sorted\_indices)), feature\_importances[sorted\_indices], align='center')

plt.yticks(range(len(sorted\_indices)), [feature\_names[i] for i in sorted\_indices])

plt.xlabel('Важность функции')

plt.ylabel('Особенность')

plt.title('Важность случайного объекта леса')

plt.show()

plot\_feature\_importances(rf\_model, X.columns)

y\_pred\_rf = rf\_model.predict(X\_test)

plt.figure(figsize=(10, 6))

plt.scatter(y\_test, y\_pred\_rf)

plt.xlabel('Правильные значения')

plt.ylabel('Прогнозируемые значения')

plt.title('Правильные vs Прогнозируемые значения (Случайный лес)')

plt.show()

errors\_rf = y\_test - y\_pred\_rf

plt.figure(figsize=(10, 6))

plt.hist(errors\_rf, bins=30)

plt.xlabel('Ошибка предсказания')

plt.ylabel('Частота')

plt.title('Распределение ошибок (Случаный лес)')

plt.show()

from sklearn.ensemble import AdaBoostRegressor

# Создание и обучение модели AdaBoost

adaboost\_model = AdaBoostRegressor(n\_estimators=300, random\_state=42)

adaboost\_model.fit(X\_train, y\_train)

# Предсказания для модели AdaBoost

y\_pred\_adaboost = adaboost\_model.predict(X\_test)

# График истинных значений и прогнозных значений для AdaBoost

plt.figure(figsize=(10, 6))

plt.scatter(y\_test, y\_pred\_adaboost)

plt.xlabel('Правильные значения')

plt.ylabel('Прогнозируемые значения')

plt.title('Пронозируемые vs Правильные значения (AdaBoost)')

plt.show()

# Гистограмма распределения ошибок прогнозирования для AdaBoost

errors\_adaboost = y\_test - y\_pred\_adaboost

plt.figure(figsize=(10, 6))

plt.hist(errors\_adaboost, bins=30)

plt.xlabel('Ошибка предсказания')

plt.ylabel('Частота')

plt.title('Распределение ошибок (AdaBoost)')

plt.show()

from sklearn.tree import plot\_tree

# Получение первого дерева из ансамбля AdaBoost

base\_tree = adaboost\_model.estimators\_[0]

# Визуализация первого дерева

plt.figure(figsize=(20,10))

plot\_tree(base\_tree, filled=True, feature\_names=X.columns)

plt.show()

from sklearn.ensemble import GradientBoostingRegressor

# Создание и обучение модели градиентного бустинга

gb\_model = GradientBoostingRegressor(n\_estimators=300, random\_state=42)

gb\_model.fit(X\_train, y\_train)

# Предсказания для модели градиентного бустинга

y\_pred\_gb = gb\_model.predict(X\_test)

# График истинных значений и прогнозных значений для градиентного бустинга

plt.figure(figsize=(10, 6))

plt.scatter(y\_test, y\_pred\_gb)

plt.xlabel('Правильные значения')

plt.ylabel('Прогнозируемые значения')

plt.title('Пронозируемые vs Правильные значения (градиентный бустинг)')

plt.show()

# Гистограмма распределения ошибок прогнозирования для градиентного бустинга

errors\_gb = y\_test - y\_pred\_gb

plt.figure(figsize=(10, 6))

plt.hist(errors\_gb, bins=30)

plt.xlabel('Ошибка предсказания')

plt.ylabel('Частота')

plt.title('Распределение ошибок (Градиентный бустинг)')

plt.show()

# Получение важности признаков из модели градиентного бустинга

feature\_importance = gb\_model.feature\_importances\_

# Создание датафрейма с важностью признаков

feature\_importance\_df = pd.DataFrame({'Feature': X.columns, 'Importance': feature\_importance})

# Сортировка признаков по важности

feature\_importance\_df = feature\_importance\_df.sort\_values(by='Importance', ascending=False)

# Визуализация важности признаков

plt.figure(figsize=(12, 8))

plt.barh(feature\_importance\_df['Feature'], feature\_importance\_df['Importance'])

plt.xlabel('Значение')

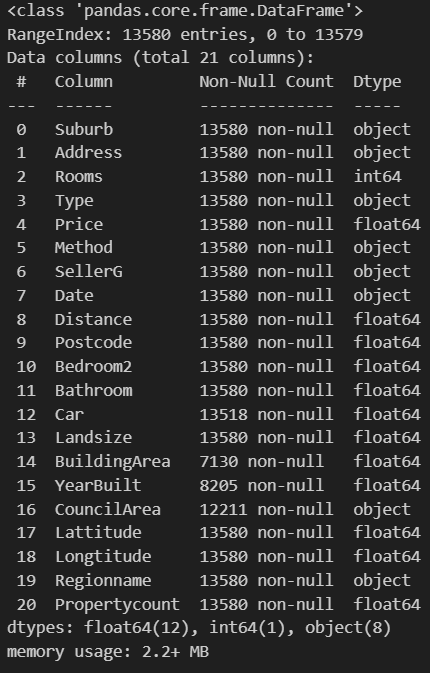
plt.ylabel('функция')

plt.title('Важность функции (Градиентный бустинг)')

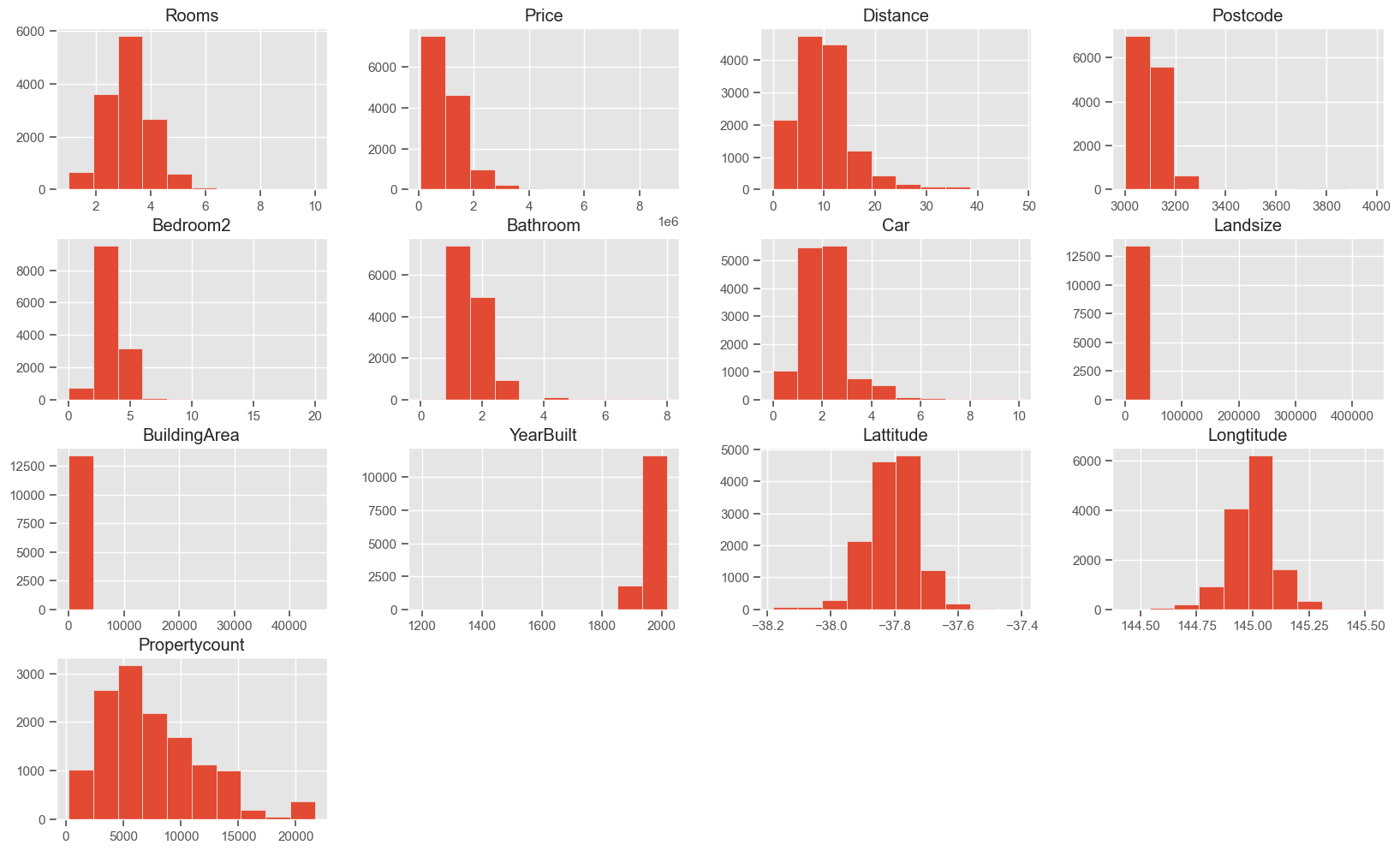
plt.show()

**Экранные формы с примерами выполнения программы**

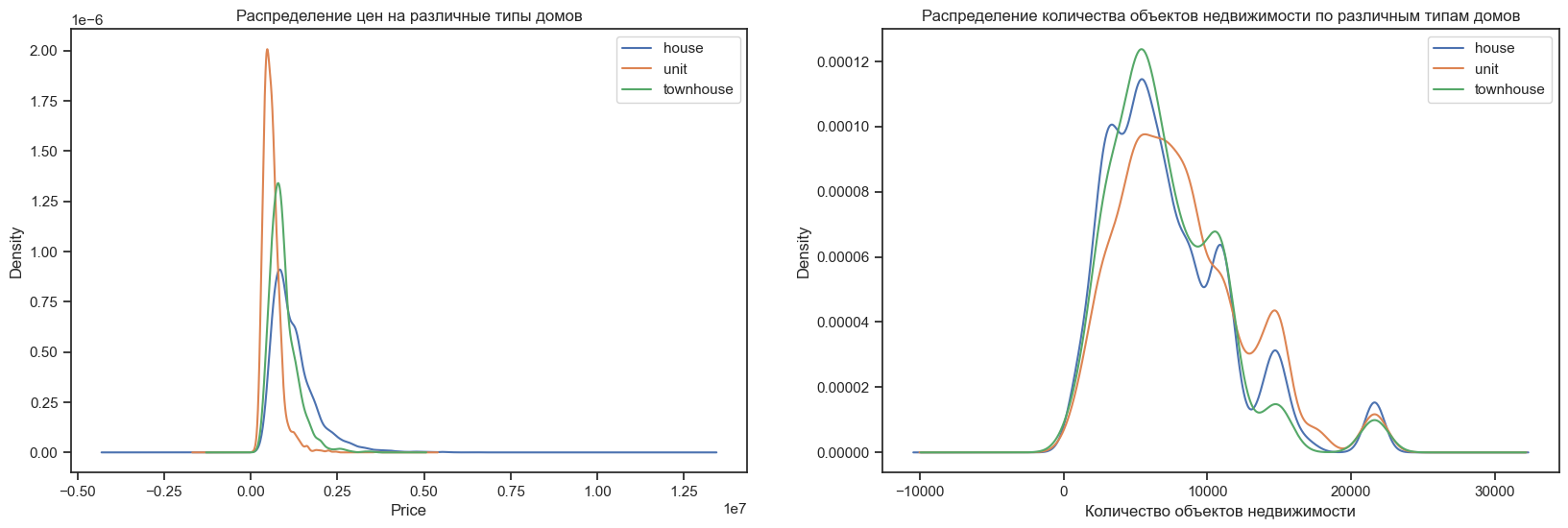
Просмотр информации о датасете



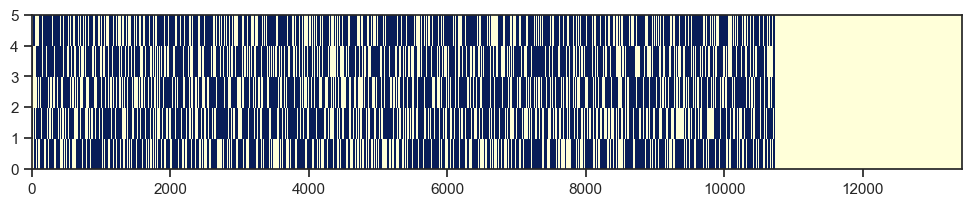
Построение гистограммы данных



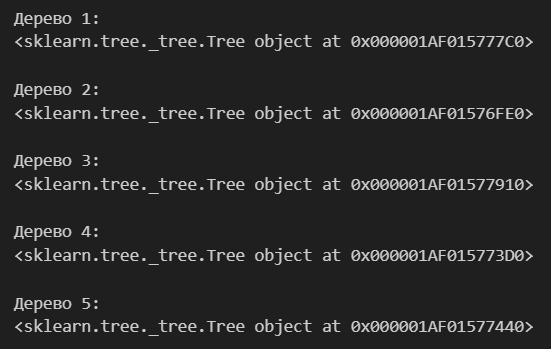
Распределение цен и количества объектов недвижимости для разных типов домов



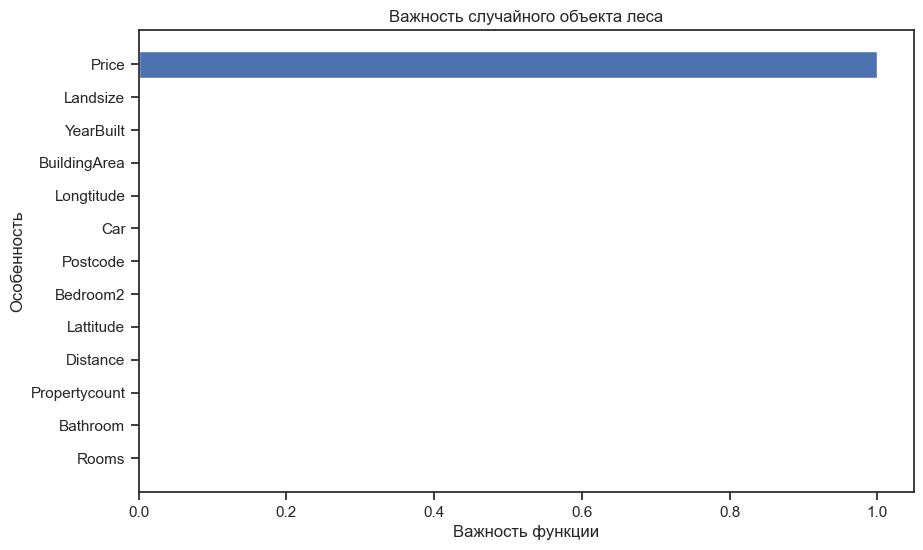
Визуализация двоичной матрицы



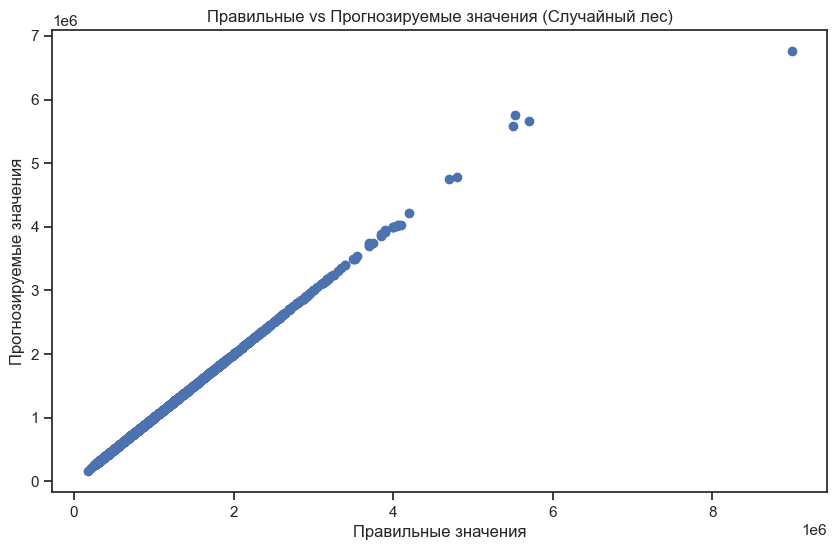
Вывод структуры каждого дерева



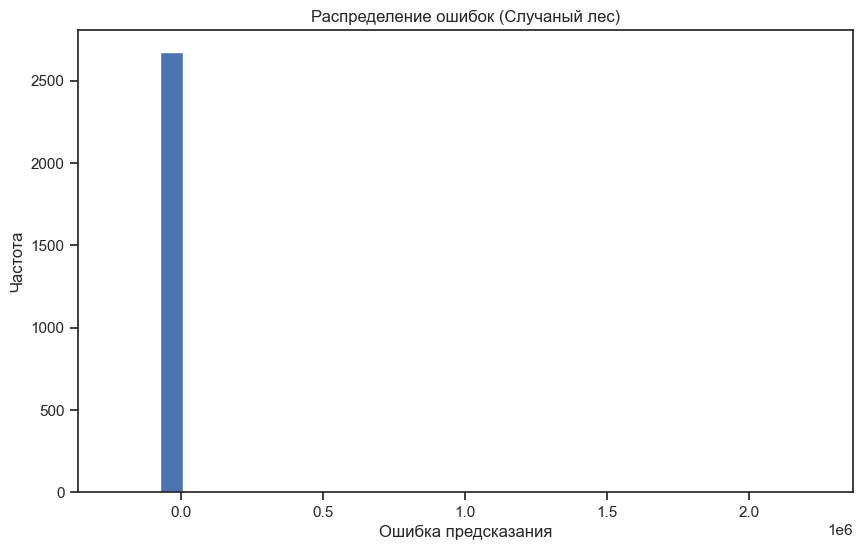
Важность признаков. Случайный лес



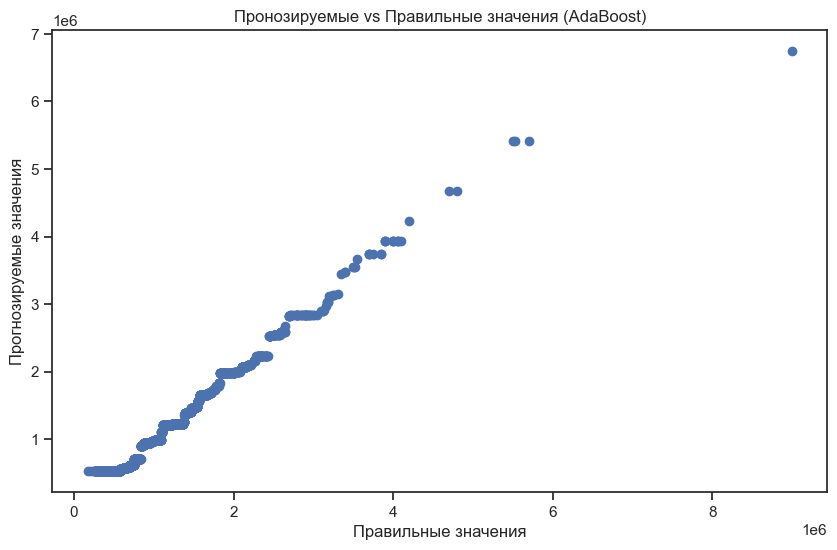
Прогнозирование

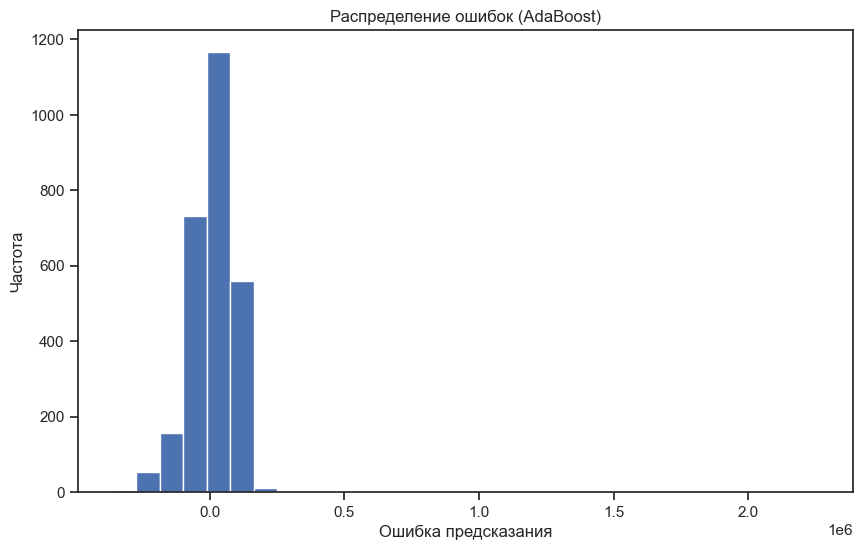


Ошибки прогнозирования

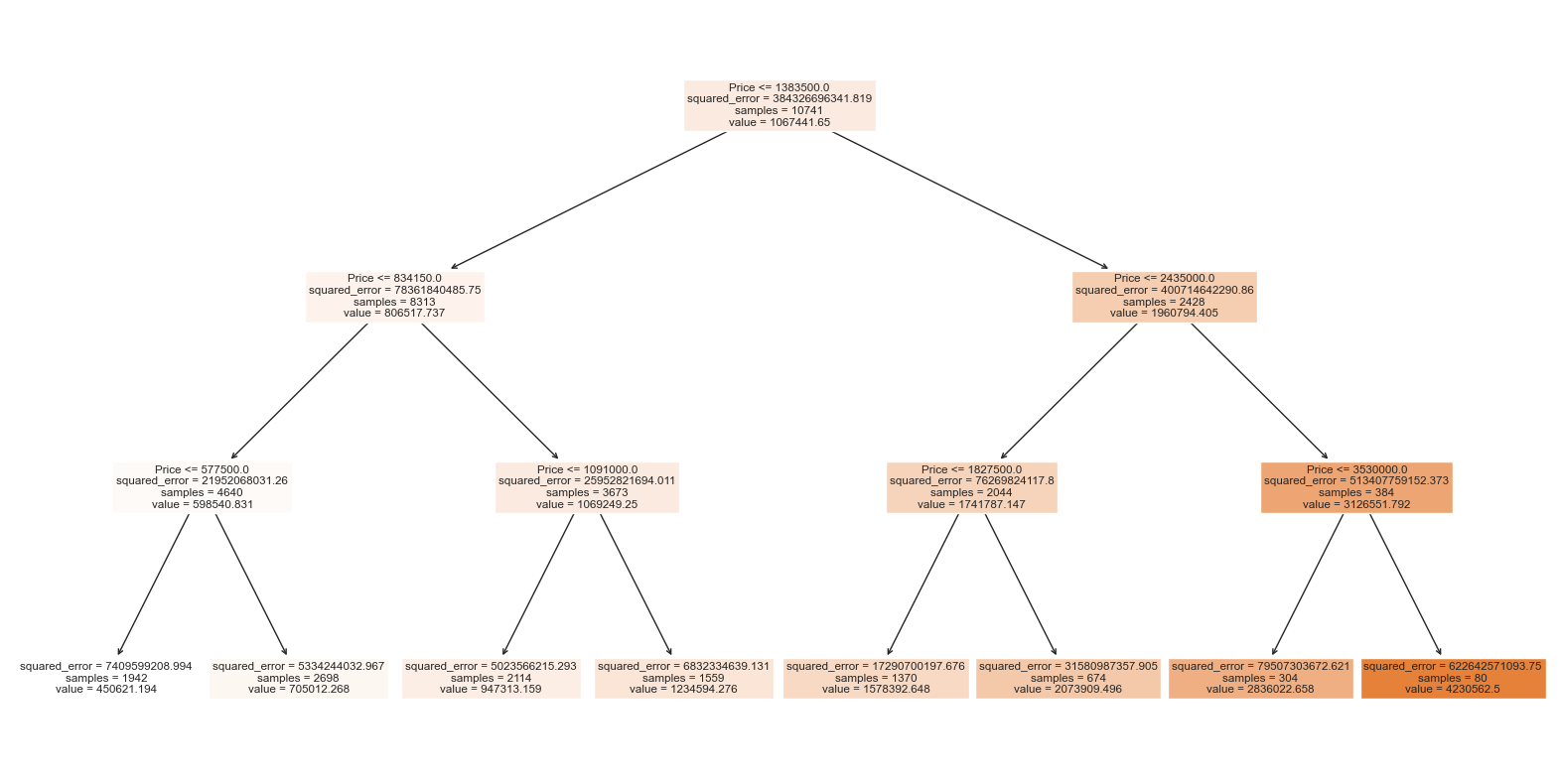


Гистограмма распределения ошибок прогнозирования для AdaBoost

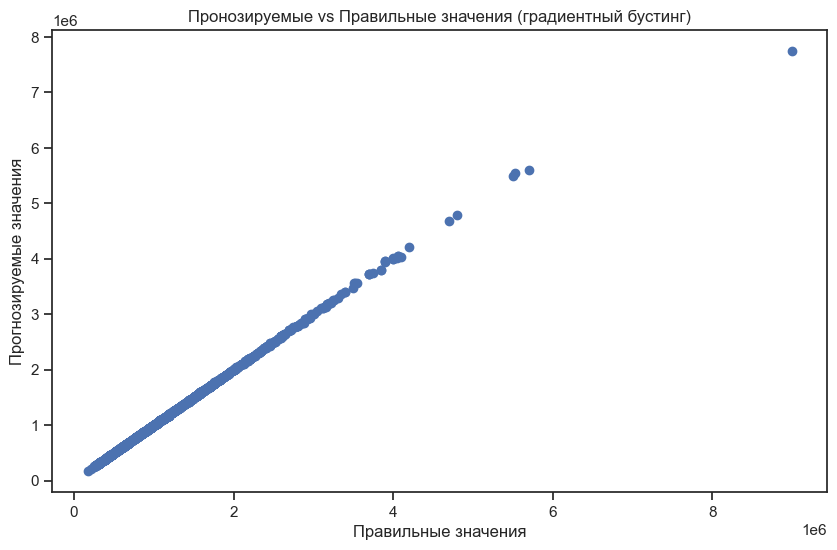


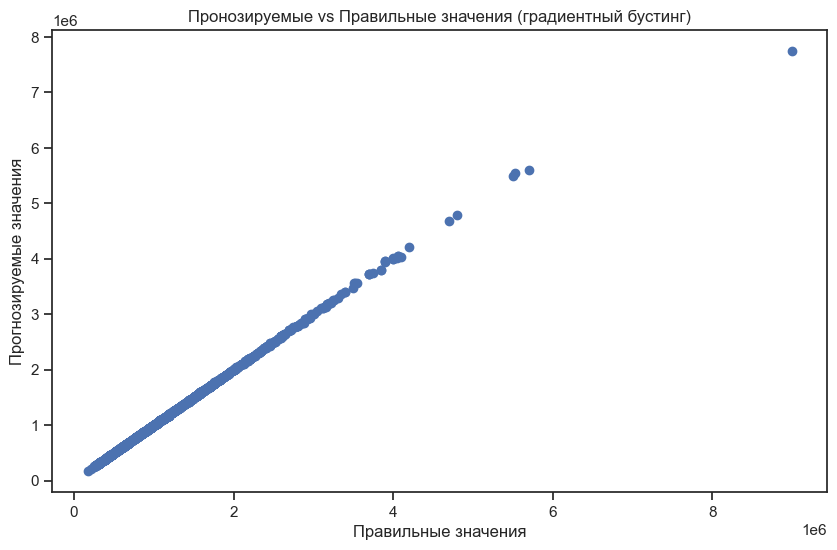


Визаулизация первого дерева



Гистограмма распределения ошибок прогнозирования для градиентного бустинга





Визуализация важности признаков

