Лабораторная работа 4

Задание

Выберите набор данных (датасет) для решения задачи классификации или регрессии.

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 модели:

одну из линейных моделей (линейную или полиномиальную регрессию при решении задачи регрессии, логистическую регрессию при решении задачи классификации);

SVM;

дерево решений.

Оцените качество моделей с помощью двух подходящих для задачи метрик. Сравните качество полученных моделей.

Постройте график, показывающий важность признаков в дереве решений.

Визуализируйте дерево решений или выведите правила дерева решений в текстовом виде.

Выбор и загрузка данных

В качестве датасета будем использовать набор данных, содержащий данные о трудоустройстве студентов. Данный набор доступен по адресу: https://www.kaggle.com/datasets/uciml/student-alcohol-consumption?select=student-por.csv

sl_no - серийный номер (номер в датасете);

gender - пол;

ssc_p - процент среднего образования;

ssc_b - Министерство образования (центральное или другое);

hsc_p - процент высшего образования;

hsc_b - Министерство образования (центральное или другое);

hsc_s - специализация полного среднего образования;

degree_p - процент выпустившихся;

degree_t - бакалавриат (сфера образования);

workex - опыт работы;

etest_p - процент теста на трудоустройство;

specialisation - специальность после выпуска;

mba_p - MBA процент;

status - статус трудоустройства (устроен или не устроен);

salary - запралата, которую предлагают кандидатам.

Импортируем библиотеки

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from google.colab import files
%matplotlib inline
sns.set(style="ticks")
```

Загружаем данные

```
files.upload()
```

Выбрать файлы StudentsP…ance.csv

- **StudentsPerformance.csv**(text/csv) - 72036 bytes, last modified: 11.10.2019 - 100% done

```
Saving StudentsPerformance.csv to StudentsPerformance.csv
{'StudentsPerformance.csv': b'"gender","race/ethnicity","parental level of education'
```

```
data = pd.read_csv('StudentsPerformance.csv')
```

Первичный анализ

Первые 5 строк датасета:

```
data.head()
```

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writi sco
0	female	group B	bachelor's degree	standard	none	72	72	

Определим размер датасета:

```
data.shape
```

```
(1000, 8)
```

Определим типы данных:

```
data.dtypes
```

```
gender                         object
race/ethnicity                 object
parental level of education    object
lunch                          object
test preparation course        object
math score                      int64
reading score                   int64
writing score                   int64
dtype: object
```

Обработка пропусков

Проверим наличие пропусков:

```
data.isnull().sum()
```

```
gender                         0
race/ethnicity                 0
parental level of education    0
lunch                          0
test preparation course        0
math score                     0
reading score                  0
writing score                  0
dtype: int64
```

Оптимизация данных

Для кодирования столбцов категорий будем использовать LabelEncoder:

```
from sklearn.preprocessing import LabelEncoder



legend = LabelEncoder()
legendarr = legend.fit_transform(data["gender"])
data["gender"] = legendarr
data = data.astype({"gender":"float"})

lerace = LabelEncoder()
leracearr = lerace.fit_transform(data["race/ethnicity"])
data["race/ethnicity"] = leracearr
data = data.astype({"race/ethnicity":"float"})

leeduc = LabelEncoder()
leeducarr = leeduc.fit_transform(data["parental level of education"])
data["parental level of education"] = leeducarr
data = data.astype({"parental level of education":"float"})

lelunch = LabelEncoder()
leluncharr = lelunch.fit_transform(data["lunch"])
data["lunch"] = leluncharr
data = data.astype({"lunch":"float"})

leprep = LabelEncoder()
lepreparr = leprep.fit_transform(data["test preparation course"])
data["test preparation course"] = lepreparr
data = data.astype({"test preparation course":"float"})
```

Проверим кодирование:

```
np.unique(legendarr), np.unique(leracearr), np.unique(leeducarr), np.unique(leluncharr), n
```

```
(array([0, 1]),
 array([0, 1, 2, 3, 4]),
 array([0, 1, 2, 3, 4, 5]),
 array([0, 1]),
 array([0, 1]))
```

И замену в датасете:

```
data.head()
```

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score
0	0.0	1.0	1.0	1.0	1.0	72	72	74

Разделение выборки на обучающую и тестовую

Разделим выборку с помощью функции train_test_split:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(data, data["writing score"], random_st
```

Размеры обучающей выборки и тестовой выборки:

```
X_train.shape, y_train.shape, X_test.shape, y_test.shape
```

```
((750, 8), (750,), (250, 8), (250,))
```

Обучение моделей

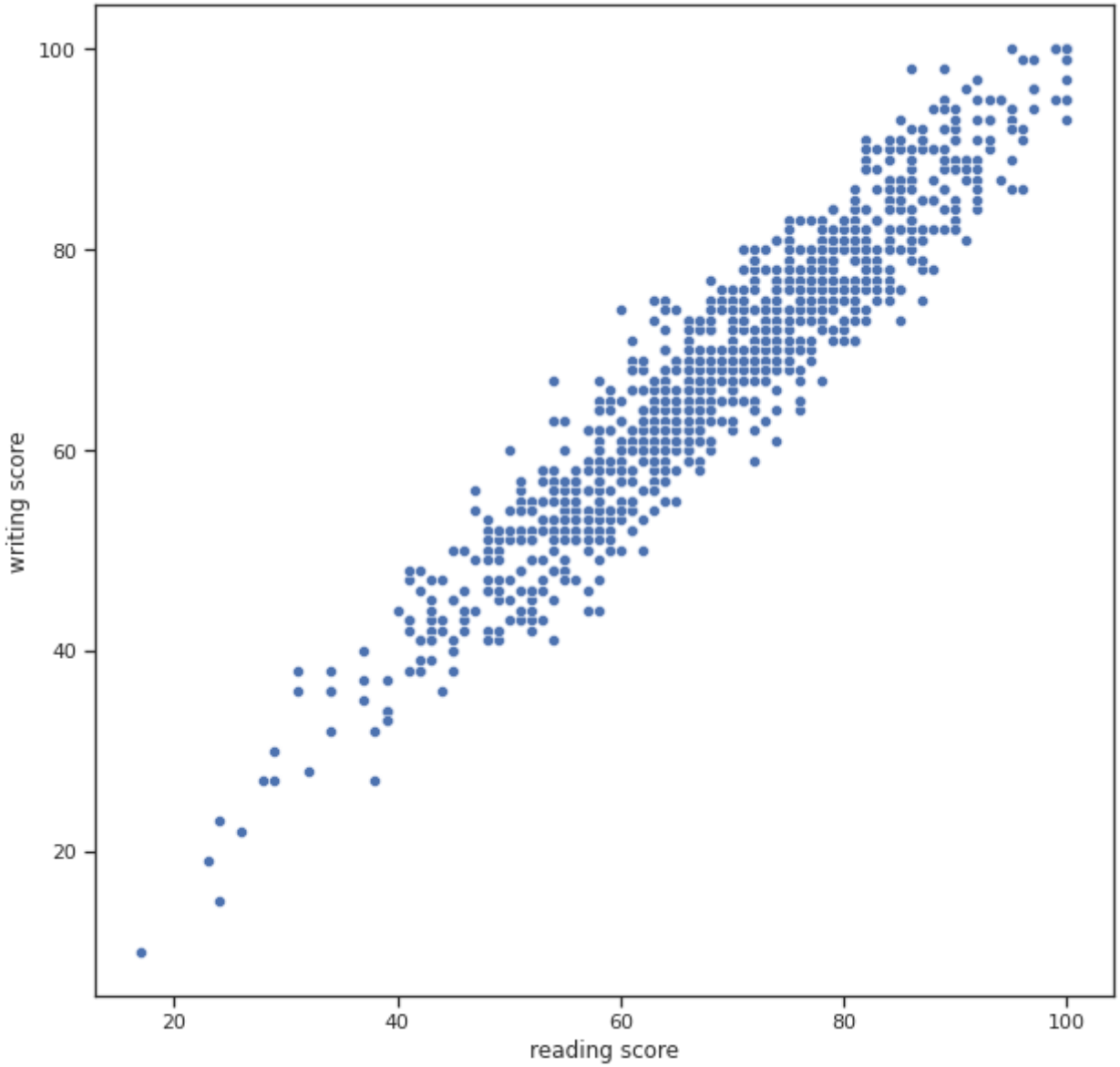
Линейная модель регрессии

```
fig, ax = plt.subplots(figsize=(15,7))
sns.heatmap(data.corr(method='pearson'), ax=ax, annot=True, fmt='.2f')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb8f093f190>
```


```
fig, ax = plt.subplots(figsize=(10,10))
sns.scatterplot(ax=ax, x='reading score', y='writing score', data=data)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb8edfa8b90>
```



Между признаками "reading score" и "writing score" прослеживается линейная зависимость (коэффициент корреляции = 0.95).

Аналитическое вычисление коэффициентов регрессии

```
from typing import Dict, Tuple


def analytic_regr_coef(x_array : np.ndarray,
                       y_array : np.ndarray) -> Tuple[float, float]:
    x_mean = np.mean(x_array)
    y_mean = np.mean(y_array)
    var1 = np.sum([(x-x_mean)**2 for x in x_array])
    cov1 = np.sum([(x-x_mean)*(y-y_mean) for x, y in zip(x_array, y_array)])
    b1 = cov1 / var1
    b0 = y_mean - b1*x_mean
    return b0, b1


x_array = data['reading score'].values
y_array = data['writing score'].values
```

Коэффициенты регрессии:

```
b0, b1 = analytic_regr_coef(x_array, y_array)
b0, b1
```

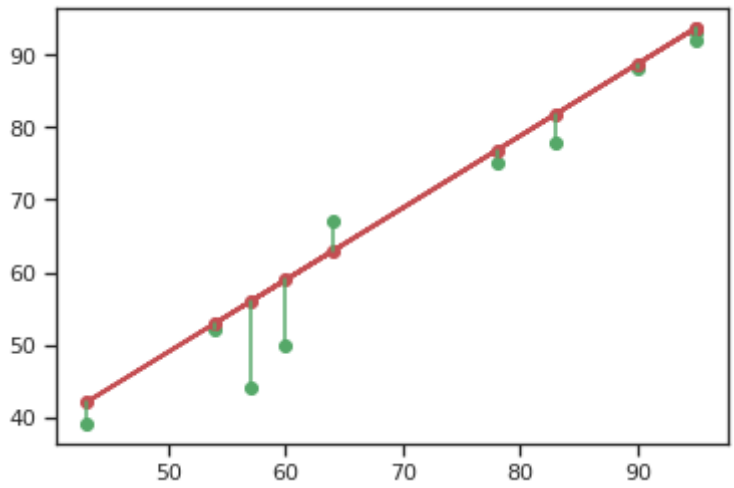
```
(-0.6675536409329368, 0.9935311142409596)
```

Отрисовка зависимости:

```
def y_regr(x_array : np.ndarray, b0: float, b1: float) -> np.ndarray:
    res = [b1*x+b0 for x in x_array]
    return res


regr_a = y_regr(x_array, b0, b1)


plt.plot(x_array, y_array, 'g.')
plt.plot(x_array, regr_a, 'b', linewidth=2.0)
plt.show()
```

Метод наименьших квадратов

```
K_mnk=10

plt.plot(x_array[1:K_mnk+1], y_array[1:K_mnk+1], 'go')
plt.plot(x_array[1:K_mnk+1], regr_a[1:K_mnk+1], '-ro', linewidth=2.0)

for i in range(len(x_array[1:K_mnk+1])):
    x1 = x_array[1:K_mnk+1][i]
    y1 = y_array[1:K_mnk+1][i]
    y2 = regr_a[1:K_mnk+1][i]
    plt.plot([x1,x1],[y1,y2],'g-')

plt.show()
```



Подбор коэффициентов через LinearRegression

```
from sklearn.linear_model import LinearRegression
```

Коэффициенты, полученные с использованием LinearRegression:

```
regr1 = LinearRegression().fit(x_array.reshape(-1, 1), y_array.reshape(-1, 1))
(b1, regr1.coef_), (b0, regr1.intercept_)
```

```
((0.9935311142409596, array([[0.99353111]])),
 (-0.6675536409329368, array([-0.66755364])))
```

Линейная модель:

```
model1 = LinearRegression()
model1.fit(X_train, y_train)
```

```
LinearRegression()
```

Чтобы изменить содержимое ячейки, дважды нажмите на нее (или выберите "Ввод")

SVM модель

```
from sklearn.svm import SVR


regr2 = SVR(kernel='linear', C=1.0)
model2 = regr2.fit(X_train, y_train)
```

Дерево решений

```
from sklearn.tree import DecisionTreeRegressor


def stat_tree(estimator):
    n_nodes = estimator.tree_.node_count
    children_left = estimator.tree_.children_left
    children_right = estimator.tree_.children_right

    node_depth = np.zeros(shape=n_nodes, dtype=np.int64)
    is_leaves = np.zeros(shape=n_nodes, dtype=bool)
    stack = [(0, -1)]  # seed is the root node id and its parent depth
    while len(stack) > 0:
        node_id, parent_depth = stack.pop()
        node_depth[node_id] = parent_depth + 1

        # If we have a test node
        if (children_left[node_id] != children_right[node_id]):
            stack.append((children_left[node_id], parent_depth + 1))
            stack.append((children_right[node_id], parent_depth + 1))
        else:
            is_leaves[node_id] = True

    print("Всего узлов:", n_nodes)
    print("Листовых узлов:", sum(is_leaves))
    print("Глубина дерева:", max(node_depth))
    print("Минимальная глубина листьев дерева:", min(node_depth[is_leaves]))
    print("Средняя глубина листьев дерева:", node_depth[is_leaves].mean())
```

Построим модель дерева с глубиной = 3:

```
regr3 = DecisionTreeRegressor(max_depth=3)
model3 = regr3.fit(X_train, y_train)
```

Выведем основную статистику для дерева:

```
stat_tree(model3)
```

```
Всего узлов: 15
Листовых узлов: 8
Глубина дерева: 3
Минимальная глубина листьев дерева: 3
Средняя глубина листьев дерева: 3.0
```

И с глубиной = 5:

```
regr4 = DecisionTreeRegressor(max_depth=5)
model4 = regr4.fit(X_train, y_train)
```

```
stat_tree(model4)
```

```
Всего узлов: 63
Листовых узлов: 32
Глубина дерева: 5
Минимальная глубина листьев дерева: 5
Средняя глубина листьев дерева: 5.0
```

Оценка качества моделей с помощью двух метрик. Сравнение качества

Оценивать качество регрессии будем при помощи двух метрик - средней абсолютной ошибки (Mean Absolute Error) и медианной абсолютной ошибки (Median Absolute Error):

```
from sklearn.metrics import mean_absolute_error, median_absolute_error
```

```
err1 = []
err2 = []
```

```
def rate_model(model):
    print("Средняя абсолютная ошибка:",
          mean_absolute_error(y_test, model.predict(X_test)))
    err1.append(mean_absolute_error(y_test, model.predict(X_test)))
    print("Медианная абсолютная ошибка:",
          median_absolute_error(y_test, model.predict(X_test)))
    err2.append(median_absolute_error(y_test, model.predict(X_test)))
```

```
rate_model(model1)
```

```
Средняя абсолютная ошибка: 4.867217739956687e-15
```

```
Медианная абсолютная ошибка: 0.0
```

```
rate_model(model2)
```

```
Средняя абсолютная ошибка: 0.03480281770536795
Медианная абсолютная ошибка: 0.0313226906224493
```

```
rate_model(model3)
```

```
Средняя абсолютная ошибка: 2.271458042303731
Медианная абсолютная ошибка: 2.1506069285324614
```

```
rate_model(model4)
```

```
Средняя абсолютная ошибка: 0.5718074060780459
Медианная абсолютная ошибка: 0.5
```

Самая лучшая по качеству по обоим метрикам - первая модель, полученная при помощи LinearRegression, а худшая - полученная с помощью дерева решений с глубиной 3.

Визуальное представление оценки:

```
fig, ax = plt.subplots(figsize=(10,10))
plt.scatter([1, 2, 3, 4], err1)
plt.scatter([1, 2, 3, 4], err2)
```

```
<matplotlib.collections.PathCollection at 0x7fb8e9f47c90>
```

График важности признаков в дереве решений

Вычисление важности признаков основано на том, какое количество раз признак встречается в условиях дерева:

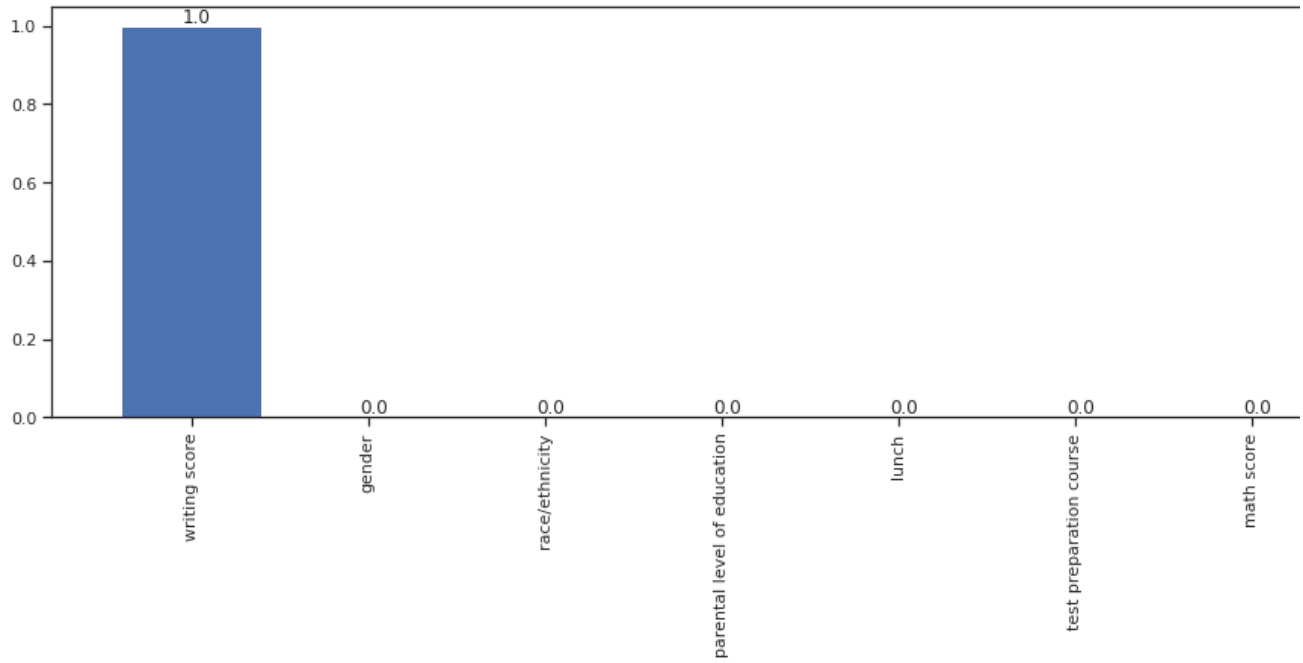
```
from operator import itemgetter

def draw_feature_importances(tree_model, X_dataset, figsize=(18,5)):
    """
    Вывод важности признаков в виде графика
    """
    # Сортировка значений важности признаков по убыванию
    list_to_sort = list(zip(X_dataset.columns.values, tree_model.feature_importances_))
    sorted_list = sorted(list_to_sort, key=itemgetter(1), reverse = True)
    # Названия признаков
    labels = [x for x,_ in sorted_list]
    # Важности признаков
    data = [x for _,x in sorted_list]
    # Вывод графика
    fig, ax = plt.subplots(figsize=figsize)
    ind = np.arange(len(labels))
    plt.bar(ind, data)
    plt.xticks(ind, labels, rotation='vertical')
    # Вывод значений
    for a,b in zip(ind, data):
        plt.text(a-0.05, b+0.01, str(round(b,3)))
    plt.show()
    return labels, data
```

Проверим для модели с глубиной = 3:

```
list(zip(data.columns.values, model3.feature_importances_))
```

```
[('gender', 0.0),
 ('race/ethnicity', 0.0),
 ('parental level of education', 0.0),
 ('lunch', 0.0),
 ('test preparation course', 0.0),
 ('math score', 0.0),
 ('reading score', 0.0),
 ('writing score', 1.0)]
```

```
stud_tree_cl_fl_1, stud_tree_cl_fd_1 = draw_feature_importances(model3, data)
```

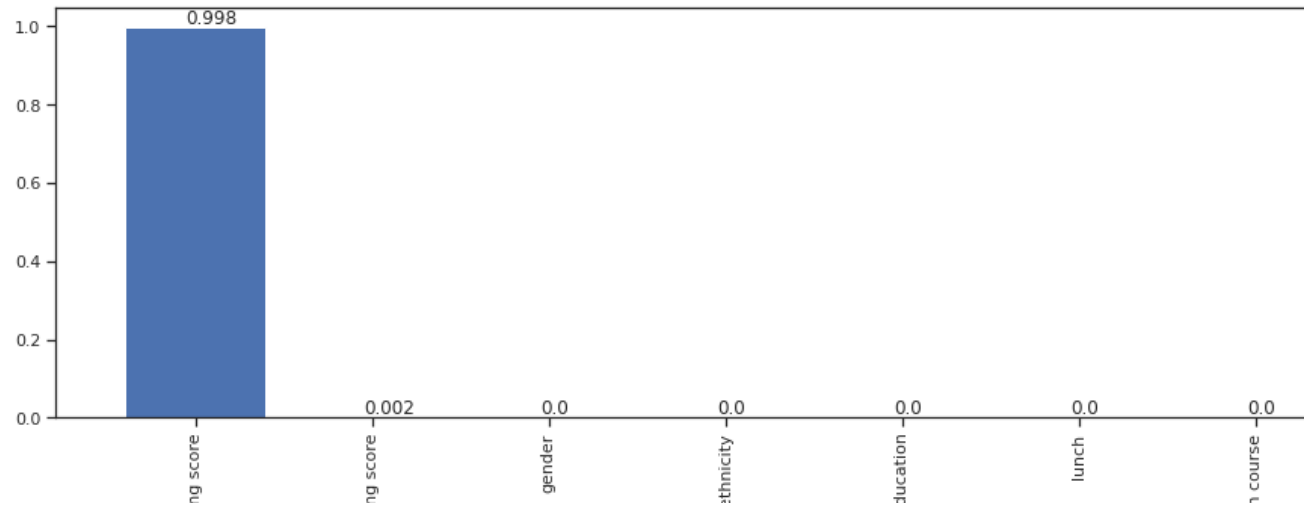


И для модели с глубиной = 5:

```
list(zip(data.columns.values, model4.feature_importances_))
```

```
[('gender', 0.0),
 ('race/ethnicity', 0.0),
 ('parental level of education', 0.0),
 ('lunch', 0.0),
 ('test preparation course', 0.0),
 ('math score', 0.0),
 ('reading score', 0.0019008910115964283),
 ('writing score', 0.9980991089884036)]
```

```
stud_tree_cl_fl_2, syud_tree_cl_fd_ = draw_feature_importances(model4, data)
```



Вывод правил дерева

Выведем правила для дерева с глубиной = 3:

```
from IPython.core.display import HTML
from sklearn.tree import export_text
tree_rules = export_text(model3, feature_names=list(data.columns))
HTML('<pre>' + tree_rules + '</pre>')
```

```
|--- writing score <= 65.50
|   |--- writing score <= 49.50
|   |   |--- writing score <= 35.00
|   |   |   |--- value: [27.31]
|   |   |--- writing score >  35.00
|   |   |   |--- value: [43.26]
|   |--- writing score >  49.50
|   |   |--- writing score <= 58.50
|   |   |   |--- value: [54.32]
|   |   |--- writing score >  58.50
|   |   |   |--- value: [62.20]
|--- writing score >  65.50
|   |--- writing score <= 80.50
|   |   |--- writing score <= 72.50
|   |   |   |--- value: [68.90]
|   |   |--- writing score >  72.50
|   |   |   |--- value: [76.44]
|   |--- writing score >  80.50
|   |   |--- writing score <= 89.50
|   |   |   |--- value: [84.39]
|   |   |--- writing score >  89.50
|   |   |   |--- value: [94.52]
```

И для дерева с глубиной = 5:

И для дерева с глубиной = 5:

```
from IPython.core.display import HTML
```

```
from sklearn.tree import export_text
tree_rules = export_text(model4, feature_names=list(data.columns))
HTML('<pre>' + tree_rules + '</pre>')
```

```
|--- writing score <= 65.50
|   |--- writing score <= 49.50
|   |   |--- writing score <= 35.00
|   |   |   |--- reading score <= 27.00
|   |   |   |   |--- writing score <= 20.50
|   |   |   |   |   |--- value: [17.00]
|   |   |   |   |--- writing score >  20.50
|   |   |   |   |   |--- value: [22.50]
|   |   |   |--- reading score >  27.00
|   |   |   |   |--- writing score <= 31.00
|   |   |   |   |   |--- value: [27.75]
|   |   |   |   |--- writing score >  31.00
|   |   |   |   |   |--- value: [33.00]
|   |   |--- writing score >  35.00
|   |   |   |--- writing score <= 43.50
|   |   |   |   |--- writing score <= 39.50
|   |   |   |   |   |--- value: [37.54]
|   |   |   |   |--- writing score >  39.50
|   |   |   |   |   |--- value: [41.93]
|   |   |   |--- writing score >  43.50
|   |   |   |   |--- writing score <= 46.50
|   |   |   |   |   |--- value: [45.41]
|   |   |   |   |--- writing score >  46.50
|   |   |   |   |   |--- value: [47.94]
|   |--- writing score >  49.50
|   |   |--- writing score <= 58.50
|   |   |   |--- writing score <= 54.50
|   |   |   |   |--- writing score <= 52.50
|   |   |   |   |   |--- value: [51.13]
|   |   |   |   |--- writing score >  52.50
|   |   |   |   |   |--- value: [53.61]
|   |   |   |--- writing score >  54.50
|   |   |   |   |--- writing score <= 56.50
|   |   |   |   |   |--- value: [55.48]
|   |   |   |   |--- writing score >  56.50
|   |   |   |   |   |--- value: [57.46]
|   |   |--- writing score >  58.50
|   |   |   |--- writing score <= 62.50
|   |   |   |   |--- writing score <= 60.50
|   |   |   |   |   |--- value: [59.65]
|   |   |   |   |--- writing score >  60.50
|   |   |   |   |   |--- value: [61.55]
|   |   |   |--- writing score >  62.50
|   |   |   |   |--- writing score <= 64.50
|   |   |   |   |   |--- value: [63.55]
|   |   |   |   |--- writing score >  64.50
|   |   |   |   |   |--- value: [65.00]
|--- writing score >  65.50
|   |--- writing score <= 80.50
|   |   |--- writing score <= 72.50
|   |   |   |--- writing score <= 68.50
|   |   |   |   |--- writing score <= 67.50
|   |   |   |   |   |--- value: [66.45]
|   |   |   |   |--- writing score >  67.50
|   |   |   |   |   |--- value: [68.00]
|   |   |   |--- writing score >  68.50
|   |   |   |   |--- writing score <= 70.50
|   |   |   |   |   |--- value: [69.58]
|   |   |   |   |--- writing score >  70.50
|   |   |   |   |   |--- value: [71.59]
```

```
|   |   |--- writing score >  72.50
|   |   |   |--- writing score <= 76.50
```

✓ 0 сек. выполнено в 16:34