|  |  |
| --- | --- |
| Изображение выглядит как эмблема, герб, нашивка, символ  Автоматически созданное описание | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное автономное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

|  |  |
| --- | --- |
| ФАКУЛЬТЕТ | ГОЛОВНОЙ УЧЕБНО-ИССЛЕДОВАТЕЛЬСКИЙ И МЕТОДИЧЕСКИЙ ЦЕНТР |
| ПРОФЕССИОНАЛЬНОЙ РЕАБИЛИТАЦИИ ЛИЦ С ОГРАНИЧЕННЫМИ |
| ВОЗМОЖНОСТЯМИ ЗДОРОВЬЯ |
| КАФЕДРА | СИСТЕМЫ ОБРАБОТКИ ИНФОРМАЦИИ И УПРАВЛЕНИЯ |

Отчёт по лабораторной работе №4 по курсу «Технологии машинного обучения».

«Линейные модели, SVM и деревья решений».

Выполнил: Проверил:

Новиков С. А. Гапанюк Ю.Е.

студент группы ИУ5-62Б

Подпись и дата: Подпись и дата:

*2025 г.*

1. **Задание лабораторной работы**

* 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 ви- де.

1. **Ячейки Jupyter-ноутбука**

# Выбор и загрузка данных

В качестве датасета будем использовать набор данных, содержащий дан- ные по продажам автомобилей в США. Данный набор доступен по адресу: [https://www.kaggle.com/datasets/gagandeep16/car-sales](http://www.kaggle.com/datasets/gagandeep16/car-sales)

Набор данных имеет следующие атрибуты:

* Manufacturer - марка
* Model - модель
* Sales\_in\_thousands - продажи в тысячах
* year\_resale\_value - годовой объем продаж
* Vehicle\_type - тип автомобиля
* Price\_in\_thousands - цена в тысячах
* Engine\_size - объем двигателя
* Horsepower - лошадиные силы
* Wheelbase - колесная база
* Width - ширина
* Length - длина
* Curb\_weight - масса
* Fuel\_capacity - топливный бак
* Fuel\_efficiency - расход топлива
* Latest\_Launch - начало производства модели
* Power\_perf\_factor - мощностной коэффициент

[1]:

## Импорт библиотек

Импортируем библиотеки с помощью команды import:

**import numpy as np import pandas as pd import seaborn as sns**

**import matplotlib.pyplot as plt**

%**matplotlib** inline xnx.xet(xtyle=DtickxD)

[2]:

## Загрузка данных

Загрузим набор данных:

data = pd.read\_cxv('Car\_xalex.cxv')

[3]:

# Первичный анализ и обработка данных

Выведем первые 5 строк датасета:

data.head()

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| [3]: | Manufacturer | Model | Salex\_in\_thouxandx | year\_rexale\_value | Vehicle\_type | \ |
| 0 | Acura | Integra | 16.919 | 16.360 | Paxxenger |  |
| 1 | Acura | TL | 39.384 | 19.875 | Paxxenger |  |
| 2 | Acura | CL | 14.114 | 18.225 | Paxxenger |  |
| 3 | Acura | RL | 8.588 | 29.725 | Paxxenger |  |
| 4 | Audi | A4 | 20.397 | 22.255 | Paxxenger |  |

Price\_in\_thouxandx Engine\_xize Horxepower Wheelbaxe Width Length \

0 21.50 1.8 140.0 101.2 67.3 172.4

1 28.40 3.2 225.0 108.1 70.3 192.9

2 NaN 3.2 225.0 106.9 70.6 192.0

3 42.00 3.5 210.0 114.6 71.4 196.6

4 23.99 1.8 150.0 102.6 68.2 178.0

Curb\_weight Fuel\_capacity Fuel\_efficiency Latext\_Launch \

0 2.639 13.2 28.0 2/2/2012

1 3.517 17.2 25.0 6/3/2011

2 3.470 17.2 26.0 1/4/2012

3 3.850 18.0 22.0 3/10/2011

4 2.998 16.4 27.0 10/8/2011

[4]:

Power\_perf\_factor

0 58.280150

1 91.370778

2 NaN

3 91.389779

4 62.777639

Определим размер датасета:

data.xhape

[4]: (157, 16)

Определим типы данных:

[5]:

data.dtypex

1. : Manufacturer object Model object

Salex\_in\_thouxandx float64

year\_rexale\_value

float64

Vehicle\_type object Price\_in\_thouxandx float64 Engine\_xize float64

Horxepower float64

Wheelbaxe float64

Width float64

Length float64

Curb\_weight float64

Fuel\_capacity float64

Fuel\_efficiency float64

Latext\_Launch object Power\_perf\_factor float64 dtype: object

1. :
2. :

data.head()

## Оптимизация данных

Удалим столбцы “Latest\_Launch” и “Vehicle\_type”, так как они не понадобятся для модели:

data = data.drop(columnx=['Latext\_Launch', 'Vehicle\_type'], axix=1)

Убедимся в верном выполнении:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [7]: | Manufacturer | Model | Salex\_in\_thouxandx | year\_rexale\_value | \ |
| 0 | Acura | Integra | 16.919 | 16.360 |  |
| 1 | Acura | TL | 39.384 | 19.875 |  |
| 2 | Acura | CL | 14.114 | 18.225 |  |
| 3 | Acura | RL | 8.588 | 29.725 |  |
| 4 | Audi | A4 | 20.397 | 22.255 |  |

Price\_in\_thouxandx Engine\_xize Horxepower Wheelbaxe Width Length \

0 21.50 1.8 140.0 101.2 67.3 172.4

1 28.40 3.2 225.0 108.1 70.3 192.9

2 NaN 3.2 225.0 106.9 70.6 192.0

3 42.00 3.5 210.0 114.6 71.4 196.6

4 23.99 1.8 150.0 102.6 68.2 178.0

Curb\_weight Fuel\_capacity Fuel\_efficiency Power\_perf\_factor

0 2.639 13.2 28.0 58.280150

1 3.517 17.2 25.0 91.370778

2 3.470 17.2 26.0 NaN

3 3.850 18.0 22.0 91.389779

4 2.998 16.4 27.0 62.777639

1. :

data.dtypex

1. : Manufacturer object Model object

Salex\_in\_thouxandx float64

year\_rexale\_value

float64

Price\_in\_thouxandx float64 Engine\_xize float64

Horxepower float64

Wheelbaxe float64

Width float64

Length float64

Curb\_weight float64

Fuel\_capacity float64

Fuel\_efficiency float64 Power\_perf\_factor float64 dtype: object

Столбцы “Manufacturer” и “Model” будем кодировать.

1. :

## Обработка пропусков

Проверим наличие пропусков:

data.ixnull().xum()

1. : Manufacturer 0

Model 0

Salex\_in\_thouxandx 0

year\_rexale\_value 36

1. :
2. :

Price\_in\_thouxandx 2

Engine\_xize 1

Horxepower 1

Wheelbaxe 1

Width 1

Length 1

Curb\_weight 2

Fuel\_capacity 1

Fuel\_efficiency 3

Power\_perf\_factor 2

dtype: int64

В датасете наблюдаются пропуски. Удалим строки с пропусками:

data = data.dropna(axix=0, how='any')

Снова проверим наличие пропусков:

data.ixnull().xum()

1. : Manufacturer 0

Model 0

Salex\_in\_thouxandx 0

year\_rexale\_value 0

Price\_in\_thouxandx 0

Engine\_xize 0

Horxepower 0

Wheelbaxe 0

Width 0

Length 0

Curb\_weight 0

Fuel\_capacity 0

Fuel\_efficiency 0

Power\_perf\_factor 0

dtype: int64

## Кодирование категориальных признаков

Для кодирования столбцов “Manufacturer” и “Model” будем использовать LabelEncoder:

1. :

**from sklearn.preprocessing import** LabelEncoder

1. :

lemanuf = LabelEncoder()

lemanufarr = lemanuf.fit\_tranxform(data[DManufacturerD]) data[DManufacturerD] = lemanufarr

data = data.axtype({DManufacturerD:DfloatD})

lemod = LabelEncoder()

lemodarr = lemod.fit\_tranxform(data[DModelD]) data[DModelD] = lemodarr

data = data.axtype({DModelD:DfloatD})

Проверим кодирование:

1. :

np.unique(lemanufarr), np.unique(lemodarr)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [14]: | (array([ 0, 1, 2, | | 3, | 4, | 5, 6, 7, | | 8, 9, | | 10, 11, 12, 13, | | | 14, | 15, 16, |
|  | 17, 18, 19, | | 20, | 21, | 22, 23, 24, | | 25]), | |  | | |  |  |
| array([ 0, | | 1, | 2, | 3, | 4, | 5, | 6, | 7, | 8, | 9, | 10, | 11, | 12, |
| 13, | | 14, | 15, | 16, | 17, | 18, | 19, | 20, | 21, | 22, | 23, | 24, | 25, |
| 26, | | 27, | 28, | 29, | 30, | 31, | 32, | 33, | 34, | 35, | 36, | 37, | 38, |
| 39, | | 40, | 41, | 42, | 43, | 44, | 45, | 46, | 47, | 48, | 49, | 50, | 51, |
| 52, | | 53, | 54, | 55, | 56, | 57, | 58, | 59, | 60, | 61, | 62, | 63, | 64, |
| 65, | | 66, | 67, | 68, | 69, | 70, | 71, | 72, | 73, | 74, | 75, | 76, | 77, |
| 78, | | 79, | 80, | 81, | 82, | 83, | 84, | 85, | 86, | 87, | 88, | 89, | 90, |
| 91, | | 92, | 93, | 94, | 95, | 96, | 97, | 98, | 99, | 100, | 101, | 102, | 103, |

1. :

104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115]))

И замену в датасете:

data.head()

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [15]: | Manufacturer | Model | Salex\_in\_thouxandx | year\_rexale\_value | \ |
| 0 | 0.0 | 62.0 | 16.919 | 16.360 |  |
| 1 | 0.0 | 107.0 | 39.384 | 19.875 |  |
| 3 | 0.0 | 88.0 | 8.588 | 29.725 |  |
| 4 | 1.0 | 4.0 | 20.397 | 22.255 |  |
| 5 | 1.0 | 5.0 | 18.780 | 23.555 |  |

Price\_in\_thouxandx Engine\_xize Horxepower Wheelbaxe Width Length \

0 21.50 1.8 140.0 101.2 67.3 172.4

1 28.40 3.2 225.0 108.1 70.3 192.9

3 42.00 3.5 210.0 114.6 71.4 196.6

4 23.99 1.8 150.0 102.6 68.2 178.0

5 33.95 2.8 200.0 108.7 76.1 192.0

Curb\_weight Fuel\_capacity Fuel\_efficiency Power\_perf\_factor

0 2.639 13.2 28.0 58.280150

1 3.517 17.2 25.0 91.370778

3 3.850 18.0 22.0 91.389779

4 2.998 16.4 27.0 62.777639

5 3.561 18.5 22.0 84.565105

## Разделение выборки на обучающую и тестовую

Разделим выборку с помощью функции train\_test\_split:

1. :

**from sklearn.model\_selection import** train\_text\_xplit

1. :

X\_train, X\_text, y\_train, y\_text = train\_text\_xplit(data, data.

*‹→* year\_rexale\_value, random\_xtate=1)

Размеры обучающей выборки и тестовой выборки:

1. :

X\_train.xhape, y\_train.xhape, X\_text.xhape, y\_text.xhape

1. : ((87, 14), (87,), (30, 14), (30,))
2. :

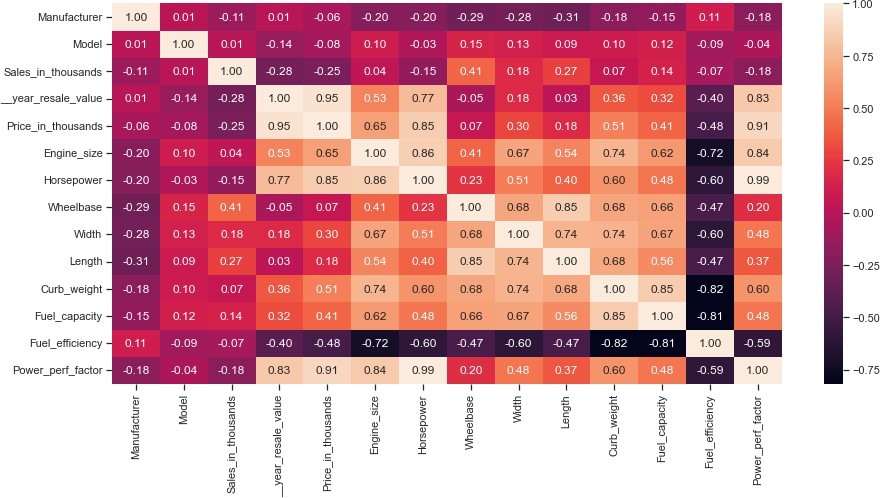
# Обучение моделей

## Линейная модель регрессии

Построим корреляционную матрицу для выявления линейной зависимости:

fig, ax = plt.xubplotx(figxize=(15,7)) xnx.heatmap(data.corr(method='pearxon'), ax=ax, annot=**True**, fmt='.2f')

1. : <AxexSubplot:>



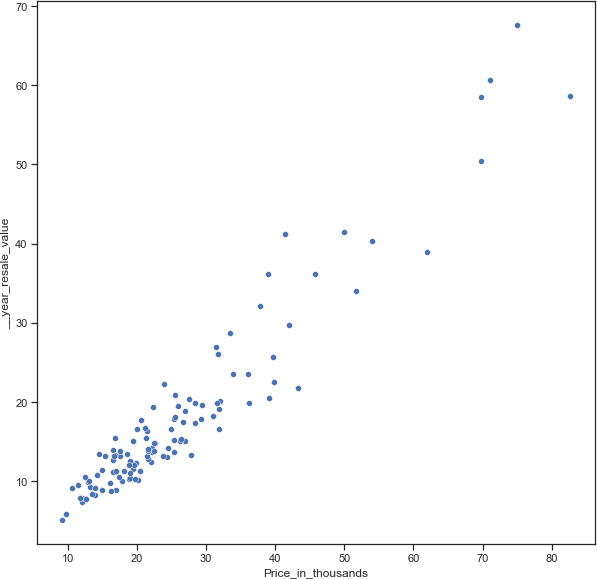
1. :

fig, ax = plt.xubplotx(figxize=(10,10))

xnx.xcatterplot(ax=ax, x='Price\_in\_thouxandx', y=' year\_rexale\_value', data=data)

1. : <AxexSubplot:xlabel='Price\_in\_thouxandx',

ylabel=' year\_rexale\_value'>



Между признаками “Price\_in\_thousands” и “year\_resale\_value” прослеживается линейная за- висимость (коэффициент корреляции = 0.95).

## Аналитическое вычисление коэффициентов регрессии

**from typing import** Dict, Tuple

1. :
2. :

**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.xum([(x-x\_mean)\*\*2 **for** x **in** x\_array])

cov1 = np.xum([(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

1. :

x\_array = data['Price\_in\_thouxandx'].valuex y\_array = data[' year\_rexale\_value'].valuex

Коэффициенты регрессии:

1. :

b0, b1 = analytic\_regr\_coef(x\_array, y\_array) b0, b1

[24]: (-2.3050406269840487, 0.7830951357632508)

Отрисовка зависимости:

1. :

**def** y\_regr(x\_array : np.ndarray, b0: float, b1: float) -> np.ndarray: rex = [b1\*x+b0 **for** x **in** x\_array]

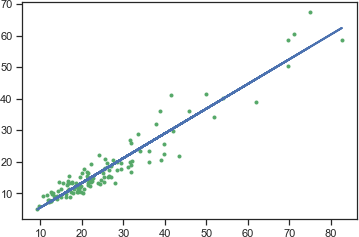
**return** rex

1. :

regr\_a = y\_regr(x\_array, b0, b1)

1. :

plt.plot(x\_array, y\_array, 'g.') plt.plot(x\_array, regr\_a, 'b', linewidth=2.0) plt.xhow()



## Метод наименьших квадратов

1. :

K\_mnk=10

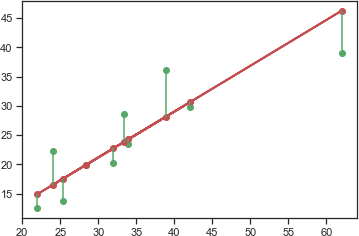
plt.plot(x\_array[1:K\_mnk+1], y\_array[1:K\_mnk+1], 'go') plt.plot(x\_array[1:K\_mnk+1], regr\_a[1:K\_mnk+1], '-ro', linewidth=2.0)

**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.xhow()



Зеленые отрезки - ошибки между истинными и предсказанными значениями.

## Решение задачи оптимизации - градиентный спуск

**from sklearn.metrics import** mean\_xquared\_error

1. :
2. :

**def** gradient\_dexcent(x\_array : np.ndarray,

y\_array : np.ndarray, b0\_0 : float,

b1\_0 : float, epochx : int,

learning\_rate : float = 0.001

) -> Tuple[float, float]: b0, b1 = b0\_0, b1\_0

k = float(len(x\_array))

**for** i **in** range(epochx):

y\_pred = b1 \* x\_array + b0

dL\_db1 = (-2/k) \* np.xum(np.multiply(x\_array, (y\_array - y\_pred))) dL\_db0 = (-2/k) \* np.xum(y\_array - y\_pred)

b1 = b1 - learning\_rate \* dL\_db1 b0 = b0 - learning\_rate \* dL\_db0

y\_pred = b1 \* x\_array + b0

**return** b0, b1, y\_pred

1. :

**def** xhow\_gradient\_dexcent(epochx, b0\_0, b1\_0):

grad\_b0, grad\_b1, grad\_y\_pred = gradient\_dexcent(x\_array, y\_array, b0\_0,

*‹→*b1\_0, epochx)

print('b0 = **{}** - (теоретический), **{}** - (градиентный спуск)'.format(b0,

*‹→*grad\_b0))

print('b1 = **{}** - (теоретический), **{}** - (градиентный спуск)'.format(b1,

*‹→*grad\_b1))

print('MSE = **{}**'.format(mean\_xquared\_error(regr\_a, grad\_y\_pred))) plt.plot(x\_array, y\_array, 'g.')

plt.plot(x\_array, regr\_a, 'b', linewidth=2.0) plt.plot(x\_array, grad\_y\_pred, 'r', linewidth=2.0) plt.xhow()

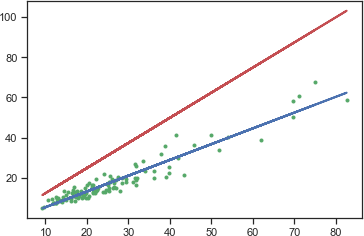
1. :

xhow\_gradient\_dexcent(1, 0, 0)

b0 = -2.3050406269840487 - (теоретический), 0.03606307692307693 - (градиентный спуск)

b1 = 0.7830951357632508 - (теоретический), 1.2474327075213676 - (градиентный спуск)

MSE = 250.15077233076002



Добьемся сходимости алгоритма путем задания других коэффициентов:

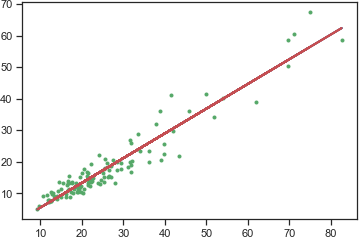
1. :

xhow\_gradient\_dexcent(100, -2.98, -22)

b0 = -2.3050406269840487 - (теоретический), -2.3021578094008817 - (градиентный спуск)

b1 = 0.7830951357632508 - (теоретический), 0.7830093489274242 - (градиентный спуск)

MSE = 1.889852950340935e-06



## Подбор коэффициентов через LinearRegression

1. :

**from sklearn.linear\_model import** LinearRegrexxion

Коэффициенты, полученные с использованием LinearRegression:

1. :

regr1 = LinearRegrexxion().fit(x\_array.rexhape(-1, 1), y\_array.rexhape(-1, 1)) (b1, regr1.coef\_), (b0, regr1.intercept\_)

1. : ((0.7830951357632508, array([[0.78309514]])),

(-2.3050406269840487, array([-2.30504063])))

Линейная модель:

1. :

model1 = LinearRegrexxion() model1.fit(X\_train, y\_train)

1. : LinearRegrexxion()

## Стохастический градиентный спуск с использованием SGDRegressor

**from sklearn.linear\_model import** SGDRegrexxor

1. :

Коэффициенты, найденные с использованием SGDRegressor:

1. :

regr2 = SGDRegrexxor().fit(x\_array.rexhape(-1, 1), y\_array) (b1, regr2.coef\_), (b0, regr2.intercept\_)

1. : ((0.7830951357632508, array([-5765028.75808579])),

(-2.3050406269840487, array([-29211796.78292314])))

Линейная модель, построенная в этом случае:

1. :

model2 = SGDRegrexxor() model2.fit(X\_train, y\_train)

1. : SGDRegrexxor()

## L1-регуляризация линейной модели

**from sklearn.linear\_model import** Laxxo

1. :

Коэффициенты, полученные с использованием Lasso:

1. :

regr3 = Laxxo().fit(x\_array.rexhape(-1, 1), y\_array) (b1, regr3.coef\_), (b0, regr3.intercept\_)

1. : ((0.7830951357632508, array([0.77805742])),

(-2.3050406269840487, -2.174213796155911))

Линейная модель:

1. :

model3 = Laxxo() model3.fit(X\_train, y\_train)

1. : Laxxo()

## SVM модель

Будем использовать класс SVR для решения задачи регрессии методом опорных векторов:

1. :
2. :

Модель SVM:

**from sklearn.svm import** SVR

regr4 = SVR(kernel='linear', C=1.0) model4 = regr4.fit(X\_train, y\_train)

## Дерево решений

1. :

**from sklearn.tree import** DecixionTreeRegrexxor

1. :
2. :

**def** xtat\_tree(extimator):

n\_nodex = extimator.tree\_.node\_count children\_left = extimator.tree\_.children\_left children\_right = extimator.tree\_.children\_right

node\_depth = np.zerox(xhape=n\_nodex, dtype=np.int64) ix\_leavex = np.zerox(xhape=n\_nodex, dtype=bool)

xtack = [(0, -1)] *# seed is the root node id and its parent depth*

**while** len(xtack) > 0:

node\_id, parent\_depth = xtack.pop() node\_depth[node\_id] = parent\_depth + 1

*# If we have a test node*

**if** (children\_left[node\_id] != children\_right[node\_id]): xtack.append((children\_left[node\_id], parent\_depth + 1)) xtack.append((children\_right[node\_id], parent\_depth + 1))

## else:

ix\_leavex[node\_id] = **True**

print(DВсего узлов:D, n\_nodex) print(DЛистовых узлов:D, xum(ix\_leavex)) print(DГлубина дерева:D, max(node\_depth))

print(DМинимальная глубина листьев дерева:D, min(node\_depth[ix\_leavex])) print(DСредняя глубина листьев дерева:D, node\_depth[ix\_leavex].mean())

Построим модель дерева с глубиной = 3:

regr5 = DecixionTreeRegrexxor(max\_depth=3) model5 = regr5.fit(X\_train, y\_train)

Выведем основную статистику для дерева:

1. :

xtat\_tree(model5)

Всего узлов: 15

Листовых узлов: 8

Глубина дерева: 3

Минимальная глубина листьев дерева: 3 Средняя глубина листьев дерева: 3.0

И с глубиной = 5:

1. :

regr6 = DecixionTreeRegrexxor(max\_depth=5) model6 = regr6.fit(X\_train, y\_train)

1. :

xtat\_tree(model6)

Всего узлов: 51

Листовых узлов: 26

Глубина дерева: 5

Минимальная глубина листьев дерева: 3

Средняя глубина листьев дерева: 4.8076923076923075

Визуализация деревьев, а также их правила представлены в пункте 6.

1. :

# Оценка качества моделей с помощью двух метрик. Сравнение качества.

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

**from sklearn.metrics import** mean\_abxolute\_error, median\_abxolute\_error

1. :

err1 = [] err2 = []

Сформируем функцию для оценивания:

1. :

**def** rate\_model(model):

print(DСредняя абсолютная ошибка:D, mean\_abxolute\_error(y\_text, model.predict(X\_text)))

err1.append(mean\_abxolute\_error(y\_text, model.predict(X\_text))) print(DМедианная абсолютная ошибка:D,

median\_abxolute\_error(y\_text, model.predict(X\_text))) err2.append(median\_abxolute\_error(y\_text, model.predict(X\_text)))

1. :

rate\_model(model1)

Средняя абсолютная ошибка: 3.4017233474514794e-14 Медианная абсолютная ошибка: 1.7763568394002505e-14

1. :

rate\_model(model2)

Средняя абсолютная ошибка: 31007083086666.18 Медианная абсолютная ошибка: 28347016356507.68

1. :

rate\_model(model3)

Средняя абсолютная ошибка: 0.11061266741068003 Медианная абсолютная ошибка: 0.08038352323471276

1. :

rate\_model(model4)

Средняя абсолютная ошибка: 0.04773643904175788 Медианная абсолютная ошибка: 0.031832622636447816

1. :

rate\_model(model5)

Средняя абсолютная ошибка: 1.4483803921568623 Медианная абсолютная ошибка: 1.372023172905526

1. :

rate\_model(model6)

Средняя абсолютная ошибка: 0.7933329365079366 Медианная абсолютная ошибка: 0.33250000000000046

1. :

Чем ближе значение средней абсолютной ошибки и медианной абсолютной ошибки к нулю, тем лучше качество регрессии.

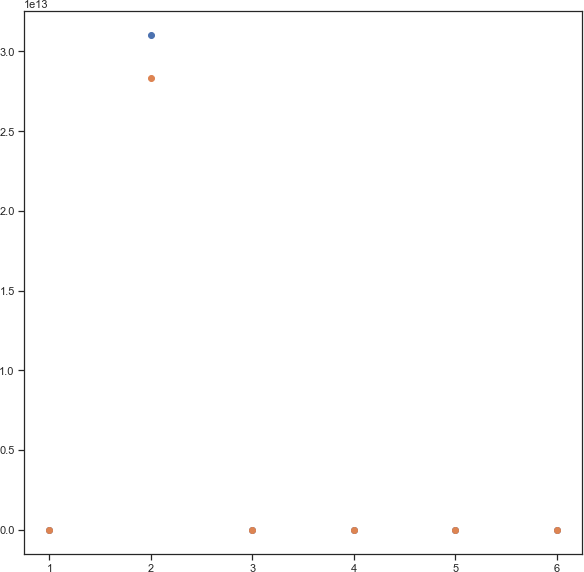
Самая лучшая по качеству по обоим метрикам - первая модель, полученная при помощи LinearRegression, а худшая - полученная через SGDRegressor.

Визуальное представление оценки:

fig, ax = plt.xubplotx(figxize=(10,10)) plt.xcatter([1, 2, 3, 4, 5, 6], err1)

plt.xcatter([1, 2, 3, 4, 5, 6], err2)

1. : <matplotlib.collectionx.PathCollection at 0x1f6c5084fa0>



1. :

err1.pop(1) err2.pop(1)

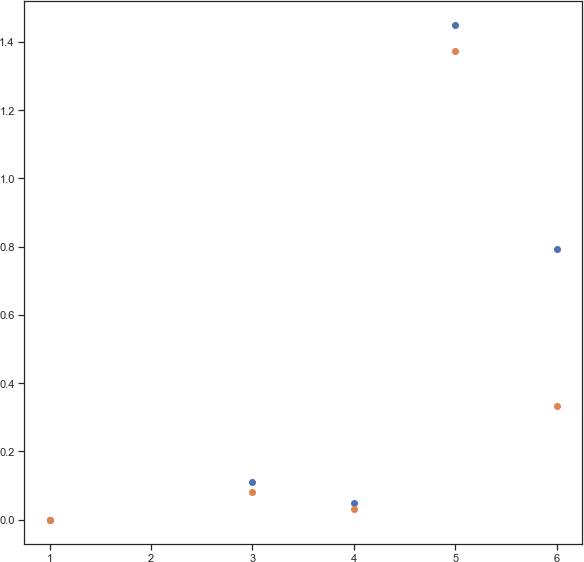
[61]: 28347016356507.68

1. :

fig, ax = plt.xubplotx(figxize=(10,10)) plt.xcatter([1, 3, 4, 5, 6], err1)

plt.xcatter([1, 3, 4, 5, 6], err2)

1. : <matplotlib.collectionx.PathCollection at 0x1f6c4fb76a0>



1. :

# График важности признаков в дереве решений

Вычисление важности признаков основано на том, какое количество раз признак встреча- ется в условиях дерева:

**from operator import** itemgetter

**def** draw\_feature\_importancex(tree\_model, X\_dataxet, figxize=(18,5)):

*"""*

*Вывод важности признаков в виде графика """*

*# Сортировка значений важности признаков по убыванию*

lixt\_to\_xort = lixt(zip(X\_dataxet.columnx.valuex, tree\_model.

*‹→*feature\_importancex\_))

xorted\_lixt = xorted(lixt\_to\_xort, key=itemgetter(1), reverxe = **True**)

*# Названия признаков*

labelx = [x **for** x,\_ **in** xorted\_lixt]

*# Важности признаков*

data = [x **for** \_,x **in** xorted\_lixt]

*# Вывод графика*

fig, ax = plt.xubplotx(figxize=figxize) ind = np.arange(len(labelx)) plt.bar(ind, data)

plt.xtickx(ind, labelx, rotation='vertical')

*# Вывод значений*

**for** a,b **in** zip(ind, data):

plt.text(a-0.05, b+0.01, xtr(round(b,3))) plt.xhow()

**return** labelx, data

Проверим для модели с глубиной = 3:

1. :

lixt(zip(data.columnx.valuex, model5.feature\_importancex\_))

1. : [('Manufacturer', 0.0),

('Model', 0.0),

('Salex\_in\_thouxandx', 0.0),

(' year\_rexale\_value',

0.9004981695348582),

('Price\_in\_thouxandx', 0.09370652359511467),

('Engine\_xize', 0.0),

('Horxepower', 0.0),

('Wheelbaxe', 0.005795306870027228),

('Width', 0.0),

('Length', 0.0),

('Curb\_weight', 0.0),

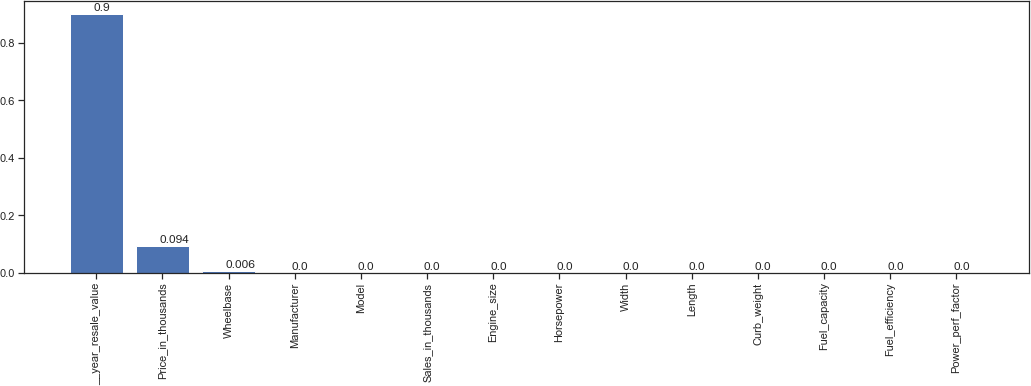
('Fuel\_capacity', 0.0),

('Fuel\_efficiency', 0.0),

('Power\_perf\_factor', 0.0)]

1. :

car\_tree\_cl\_fl\_1, car\_tree\_cl\_fd\_1 = draw\_feature\_importancex(model5, data)



И для модели с глубиной = 5:

1. :

lixt(zip(data.columnx.valuex, model6.feature\_importancex\_))

1. : [('Manufacturer', 0.0),

('Model', 0.0),

('Salex\_in\_thouxandx', 0.005686857877003804),

(' year\_rexale\_value',

0.8976880644389448),

('Price\_in\_thouxandx', 0.09396949590123833),

('Engine\_xize', 0.00020496267968645642),

('Horxepower', 0.00020093402747307438),

('Wheelbaxe', 0.0),

('Width', 0.0015354638102150987), ('Length', 7.859870478468979e-05),

('Curb\_weight', 0.0),

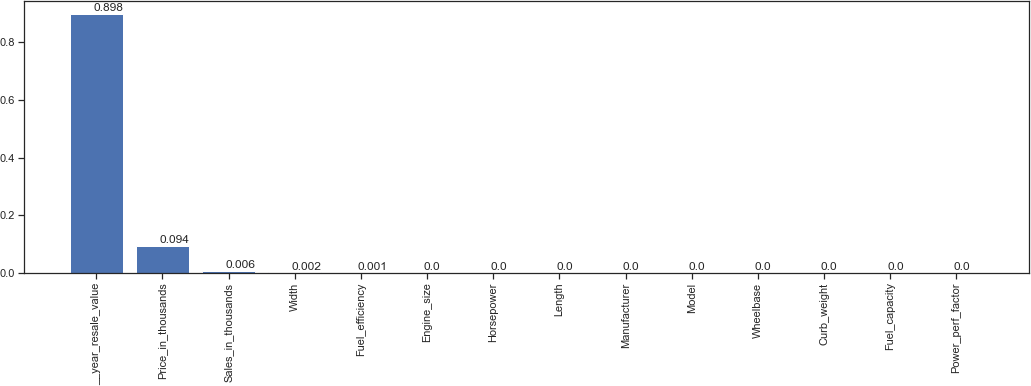
('Fuel\_capacity', 0.0),

('Fuel\_efficiency', 0.0006356225606536772),

('Power\_perf\_factor', 0.0)]

1. :

car\_tree\_cl\_fl\_2, car\_tree\_cl\_fd\_ = draw\_feature\_importancex(model6, data)



# Визуализация деревьев решений и его правила

1. :

**from io import** StringIO

**from IPython.display import** Image

**import graphviz import pydotplus**

**from sklearn.tree import** export\_graphviz

## Визуализация деревьев

1. :



**def** get\_png\_tree(tree\_model\_param, feature\_namex\_param): dot\_data = StringIO() export\_graphviz(tree\_model\_param, out\_file=dot\_data,

*‹→*feature\_namex=feature\_namex\_param,

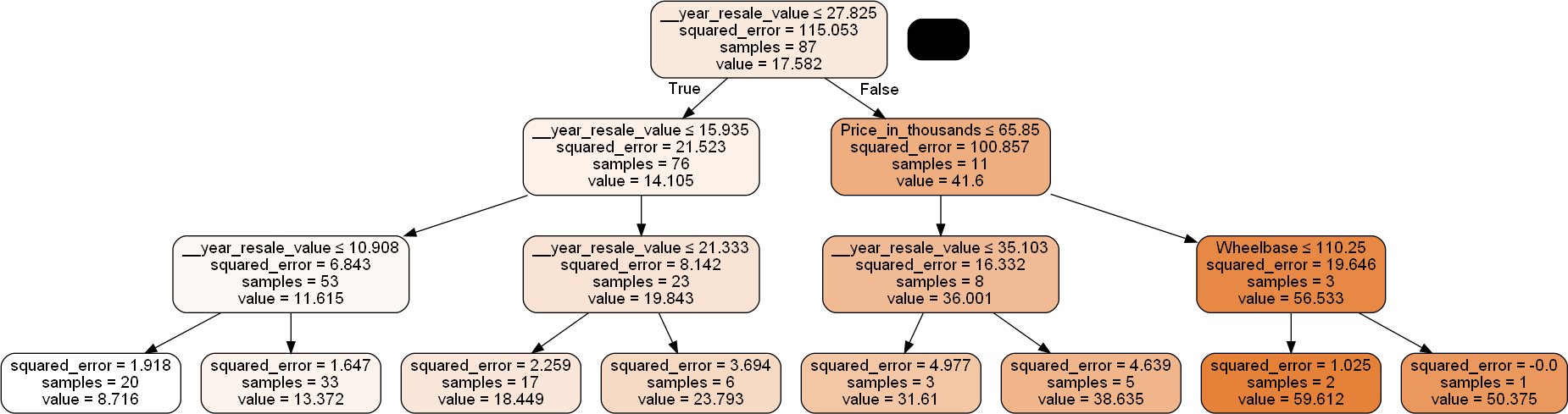
filled=**True**, rounded=**True**, xpecial\_characterx=**True**) graph = pydotplux.graph\_from\_dot\_data(dot\_data.getvalue())

**return** graph.create\_png()

Визуализируем дерево с глубиной = 3:

1. :

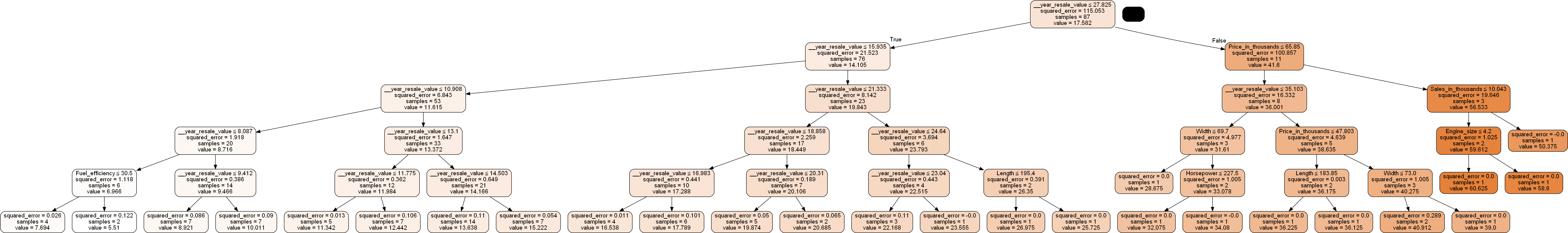
Image(get\_png\_tree(model5, data.columnx), height='100%')

[70]:

И с глубиной = 5:

1. :

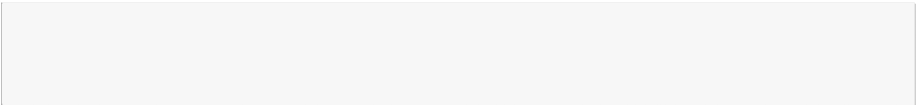
Image(get\_png\_tree(model6, data.columnx), height='100%')

[71]:

## Вывод правил дерева

Выведем правила для дерева с глубиной = 3:

In [74]:



**from IPython.core.display import** HTML

**from sklearn.tree import** export\_text

tree\_rules = export\_text(model5, feature\_names=list(data.columns)) HTML('<pre>' + tree\_rules + '</pre>')

Out[74]:

|--- year\_resale\_value <= 27.82

| |--- year\_resale\_value <= 15.94

| | |--- year\_resale\_value <= 10.91

| | | |--- value: [8.72]

| | |--- year\_resale\_value > 10.91

| | | |--- value: [13.37]

| |--- year\_resale\_value > 15.94

| | |--- year\_resale\_value <= 21.33

| | | |--- value: [18.45]

| | |--- year\_resale\_value > 21.33

| | | |--- value: [23.79]

|--- year\_resale\_value > 27.82

| |--- year\_resale\_value <= 45.91

| | |--- year\_resale\_value <= 35.10

| | | |--- value: [31.61]

| | |--- year\_resale\_value > 35.10

| | | |--- value: [38.64]

**|** |--- year\_resale\_value > 45.91

| | |--- year\_resale\_value <= 54.49

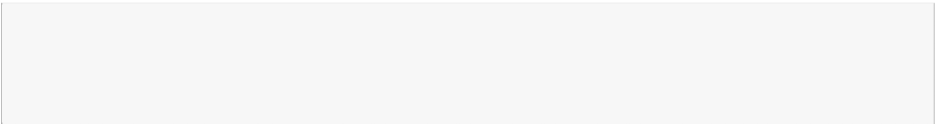
| | | |--- value: [50.38]

| | |--- year\_resale\_value > 54.49

| | | |--- value: [59.61]

И для дерева с глубиной = 5:

In [75]:



**from IPython.core.display import** HTML

**from sklearn.tree import** export\_text

tree\_rules = export\_text(model6, feature\_names=list(data.columns)) HTML('<pre>' + tree\_rules + '</pre>')

Out[75]:

|--- year\_resale\_value <= 27.82

| |--- year\_resale\_value <= 15.94

| | |--- year\_resale\_value <= 10.91

| | | |--- year\_resale\_value <= 8.09

| | | | |--- Fuel\_efficiency <= 30.50

| | | | | |--- value: [7.69]

| | | | |--- Fuel\_efficiency > 30.50

| | | | | |--- value: [5.51]

| | | |--- year\_resale\_value > 8.09

| | | | |--- year\_resale\_value <= 9.41

| | | | | |--- value: [8.92]

| | | | |--- year\_resale\_value > 9.41

| | | | | |--- value: [10.01]

| | |--- year\_resale\_value > 10.91

| | | |--- year\_resale\_value <= 13.10

| | | | |--- year\_resale\_value <= 11.77

| | | | | |--- value: [11.34]

| | | | |--- year\_resale\_value > 11.77

| | | | | |--- value: [12.44]

| | | |--- year\_resale\_value > 13.10

| | | | |--- year\_resale\_value <= 14.50

| | | | | |--- value: [13.64]

| | | | |--- year\_resale\_value > 14.50

| | | | | |--- value: [15.22]

| |--- year\_resale\_value > 15.94

| | |--- year\_resale\_value <= 21.33

| | | |--- year\_resale\_value <= 18.86

| | | | |--- year\_resale\_value <= 16.98

| | | | | |--- value: [16.54]

| | | | |--- year\_resale\_value > 16.98

| | | | | |--- value: [17.79]

| | | |--- year\_resale\_value > 18.86

| | | | |--- year\_resale\_value <= 20.31

| | | | | |--- value: [19.87]

| | | | |--- year\_resale\_value > 20.31

| | | | | |--- value: [20.69]

| | |--- year\_resale\_value > 21.33

| | | |--- year\_resale\_value <= 24.64

| | | | |--- year\_resale\_value <= 23.04

| | | | | |--- value: [22.17]

| | | | |--- year\_resale\_value > 23.04

| | | | | |--- value: [23.55]

| | | |--- year\_resale\_value > 24.64

| | | | |--- Fuel\_capacity <= 18.75

| | | | | |--- value: [26.98]

| | | | |--- Fuel\_capacity > 18.75

| | | | | |--- value: [25.73]

|--- year\_resale\_value > 27.82

| |--- year\_resale\_value <= 45.91

| | |--- year\_resale\_value <= 35.10

|--- Curb\_weight <= 3.42

|  |  |  |
| --- | --- | --- |
| | | | | | |
| | | | | | |
| | | | | | |

| |--- value: [28.68]

|--- Curb\_weight > 3.42

| | | | |--- Horsepower <= 227.50

| | | | | |--- value: [32.08]

| | | | |--- Horsepower > 227.50

| | | | | |--- value: [34.08]

| | |--- year\_resale\_value > 35.10

| | | |--- Curb\_weight <= 3.65

|  |  |  |  |
| --- | --- | --- | --- |
| | | | | | | |--- Curb\_weight <= 3.65 |
| | | | | | | | |--- Manufacturer <= 3.50 |
| | | | | | | | | |--- value: [36.12] |
| | | | | | | | |--- Manufacturer > 3.50 |
| | | | | | | | | |--- value: [36.23] |
| | | | | | | |--- Curb\_weight > 3.65 |
| | | | | | | | |--- Model <= 24.50 |
| | | | | | | | | |--- value: [39.00] |
| | | | | | | | |--- Model > 24.50 |
| | | | | | | | | |--- value: [40.91] |
| | | |--- year\_resale\_value > 45.91 | | |
| | | | |--- Curb\_weight <= 4.13 | | |
| | | | | |--- Model <= 60.50 | | |
| | | | | | |--- value: [60.62] | | |
| | | | | |--- Model > 60.50 | | |
| | | | | | |--- value: [58.60] | | |
| | | | |--- Curb\_weight > 4.13 | | |
| | | | | |--- value: [50.38] | | |