



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:
Новиков С. А.
студент группы ИУ5-62Б

Проверил:
Гапанюк Ю.Е.

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 виде.

2. Ячейки Jupyter-ноутбука

2.1. Выбор и загрузка данных

В качестве датасета будем использовать набор данных, содержащий данные по продажам автомобилей в США. Данный набор доступен по адресу: <https://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

2.1.1. Импорт библиотек

Импортируем библиотеки с помощью команды `import`:

```
[1]: import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
sns.set(xstyle='D', ystyle='D')
```

2.1.2. Загрузка данных

Загрузим набор данных:

```
[2]: data = pd.read_csv('Car_xalex.csv')
```

2.2. Первичный анализ и обработка данных

Выведем первые 5 строк датасета:

```
[3]: 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

      Power_perf_factor
0          58.280150
1          91.370778
2                 NaN
3          91.389779
4          62.777639
```

Определим размер датасета:

```
[4]: data.xshape
```

```
[4]: (157, 16)
```

Определим типы данных:

```
[5]: data.dtypes
```

```
[5]: Manufacturer      object
      Model           object
      Salex_in_thouxandx  float64
      __year_rexale_value  float64
```

Vehicle_type	object
Price_in_thouxandx	float64
Engine_xize	float64

```

Horsepower      float64
Wheelbase       float64
Width           float64
Length          float64
Curb_weight     float64
Fuel_capacity   float64
Fuel_efficiency float64
Latest_Launch   object
Power_perf_factor float64
dtype: object

```

2.2.1. Оптимизация данных

Удалим столбцы "Latest_Launch" и "Vehicle_type", так как они не понадобятся для модели:

```
[6]: data = data.drop(columnx=['Latest_Launch', 'Vehicle_type'], axis=1)
```

Убедимся в верном выполнении:

```
[7]: data.head()
```

```

[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  Horsepower  Wheelbase  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

```

```
[8]: data.dtypes
```

```

[8]:  Manufacturer      object
      Model           object
      Salex_in_thouxandx  float64
      __year_rexale_value  float64

      Price_in_thouxandx  float64
      Engine_xize        float64
      Horsepower         float64
      Wheelbase          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.

2.2.2. Обработка пропусков

Проверим наличие пропусков:

```
[9] : data.isnull().sum()
```

```
[9] : Manufacturer      0
      Model             0
      Salex_in_thouxandx 0
      __year_rexale_value 36
      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:

```
[10] : data = data.dropna(axix=0, how='any')
```

Снова проверим наличие пропусков:

```
[11] : data.isnull().sum()
```

```
[11] : 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
```

2.2.3. Кодирование категориальных признаков

Для кодирования столбцов "Manufacturer" и "Model" будем использовать LabelEncoder:


```
[12] : from sklearn.preprocessing import LabelEncoder
```

```
[13] : lemanuf = LabelEncoder()
      lemanufarr = lemanuf.fit_transform(data[DManufacturerD])
      data[DManufacturerD] = lemanufarr
      data = data.axtype({DManufacturerD:DfloatD})

      lemod = LabelEncoder()
      lemodarr = lemod.fit_transform(data[DModelID])
      data[DModelID] = lemodarr
      data = data.axtype({DModelID:DfloatD})
```

Проверим кодирование:

```
[14] : 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,
        104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115]))
```

И замену в датасете:

```
[15] : 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	Horexpower	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

2.2.4. Разделение выборки на обучающую и тестовую

Разделим выборку с помощью функции `train_test_split`:

```
[16] : from sklearn.model_selection import train_test_split
```

```
[17] : X_train, X_test, y_train, y_test = train_test_split(data, data.
    ↪ __year_rexale_value, random_xtate=1)
```

Размеры обучающей выборки и тестовой выборки:

```
[18] : X_train.xhape, y_train.xhape, X_test.xhape, y_test.xhape
```

```
[18] : ((87, 14), (87,), (30, 14), (30,))
```

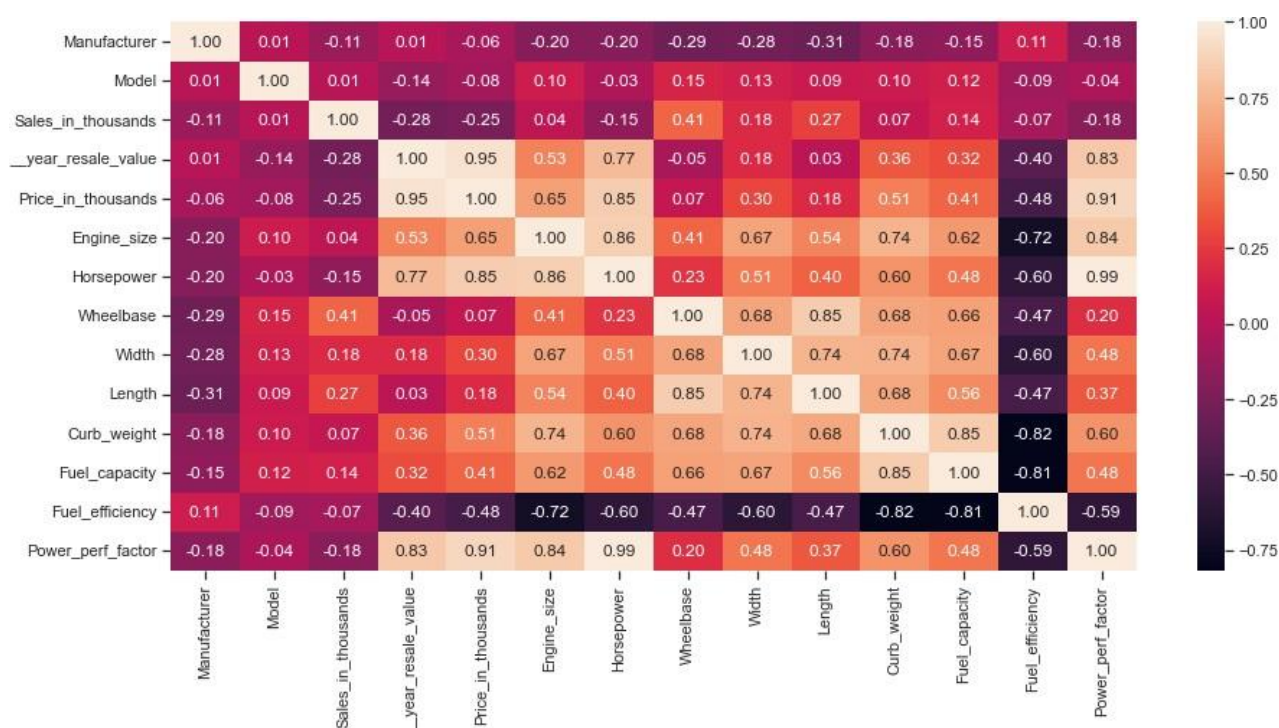
2.3. Обучение моделей

2.3.1. Линейная модель регрессии

Построим корреляционную матрицу для выявления линейной зависимости:

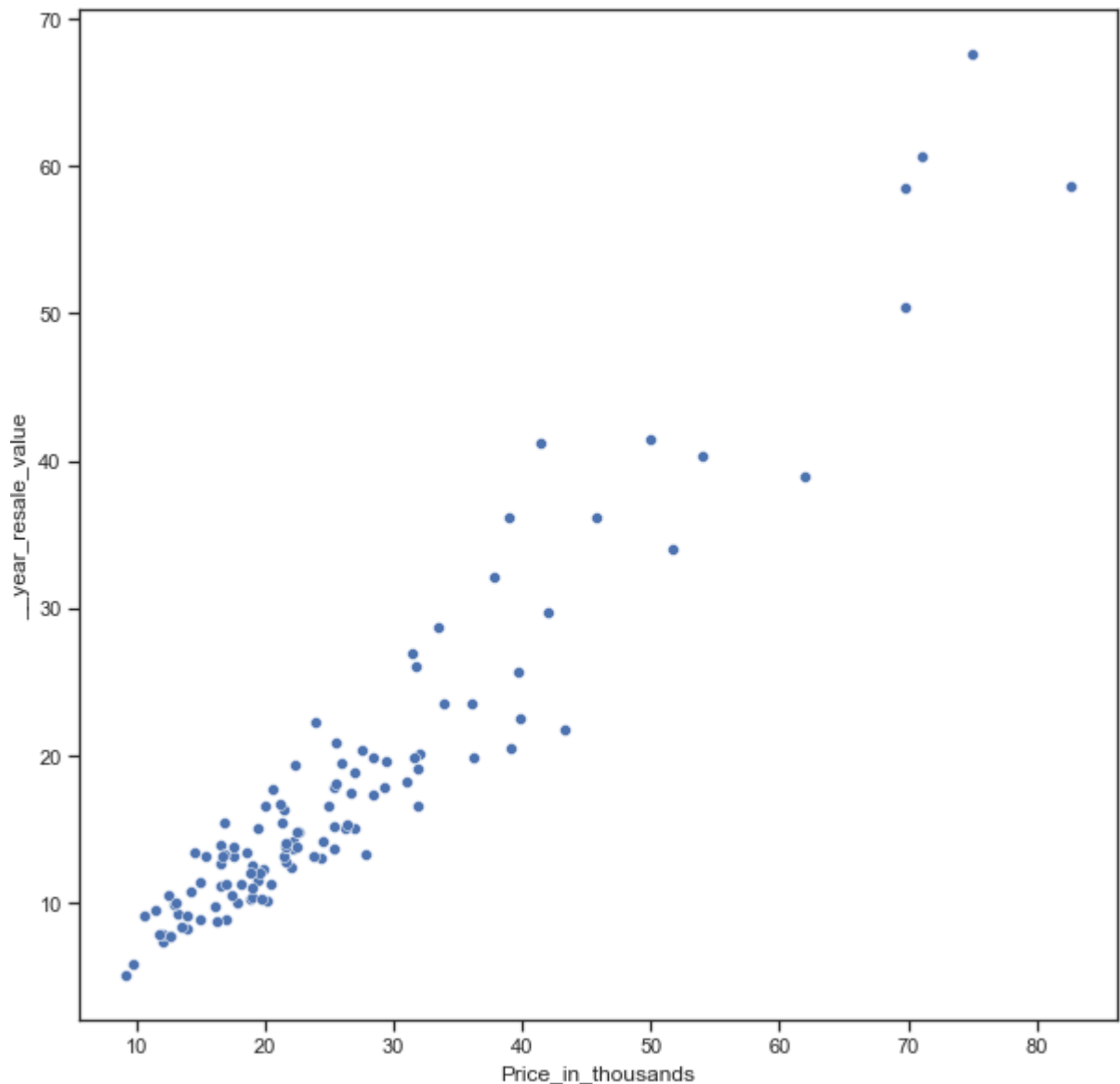
```
[19] : fig, ax = plt.xubplotx(figsize=(15,7))
    xnx.heatmap(data.corr(method='pearxon'), ax=ax, annot=True, fmt='.2f')
```

```
[19] : <AxexSubplot:>
```



```
[20] : fig, ax = plt.xubplotx(figsize=(10,10))
    xnx.xcatterplot(ax=ax, x='Price_in_thouxandx', y='__year_rexale_value', data=data)
```

```
[20] : <AxexSubplot:xlabel='Price_in_thouxandx', ylabel='__year_rexale_value'>
```



Между признаками “Price_in_thousands” и “year_resale_value” прослеживается линейная зависимость (коэффициент корреляции = 0.95).

Аналитическое вычисление коэффициентов регрессии

```
[21] : from typing import Dict, Tuple
```

```
[22] : 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
```

```
[23] : x_array = data['Price_in_thouxandx'].valuex
       y_array = data['__year_rexale_value'].valuex
```

Коэффициенты регрессии:

```
[24] : b0, b1 = analytic_regr_coef(x_array, y_array)
       b0, b1
```

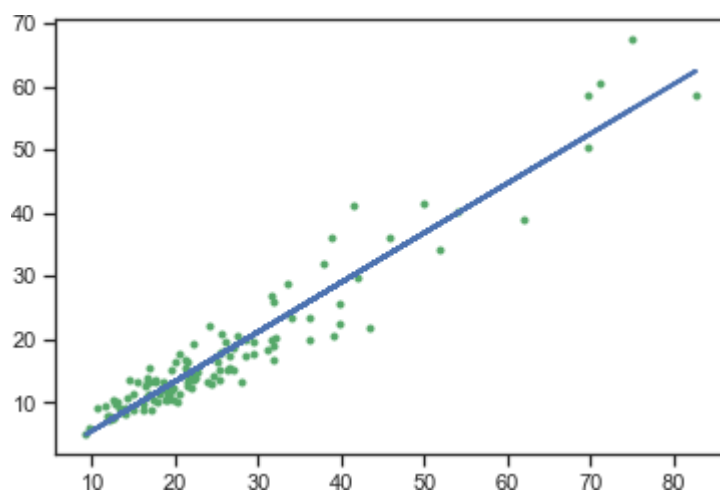
```
[24]: (-2.3050406269840487, 0.7830951357632508)
```

Отрисовка зависимости:

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

```
[26] : regr_a = y_regr(x_array, b0, b1)
```

```
[27] : plt.plot(x_array, y_array, 'g. ')
       plt.plot(x_array, regr_a, 'b', linewidth=2.0)
       plt.xshow()
```



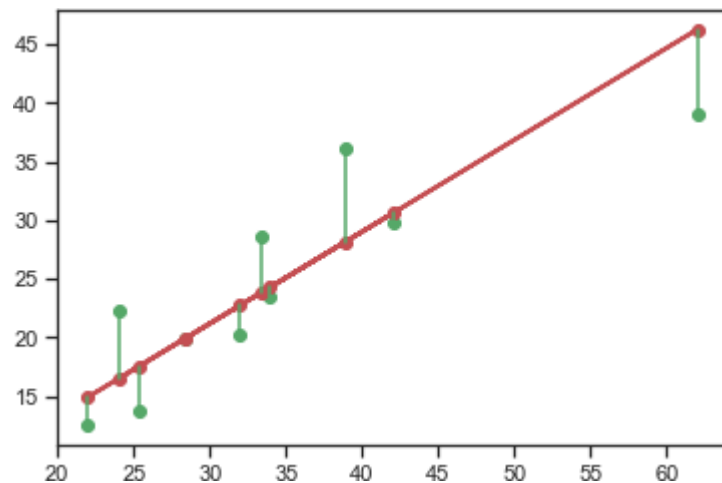
Метод наименьших квадратов

```
[28] : 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.xshow()
```



Зеленые отрезки - ошибки между истинными и предсказанными значениями.

Решение задачи оптимизации - градиентный спуск

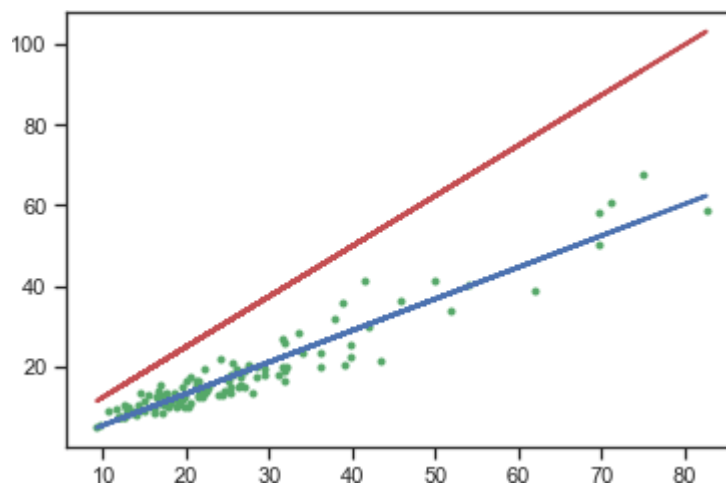
```
[29] : from sklearn.metrics import mean_squared_error
```

```
[30] : def gradient_descent(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.sum(np.multiply(x_array, (y_array - y_pred)))
        dL_db0 = (-2/k) * np.sum(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
```

```
[31] : def xshow_gradient_descent(epochx, b0_0, b1_0):
    grad_b0, grad_b1, grad_y_pred = gradient_descent(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_squared_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.show()
```

[32] : `xhgw_gradient_descent(1, 0, 0)`

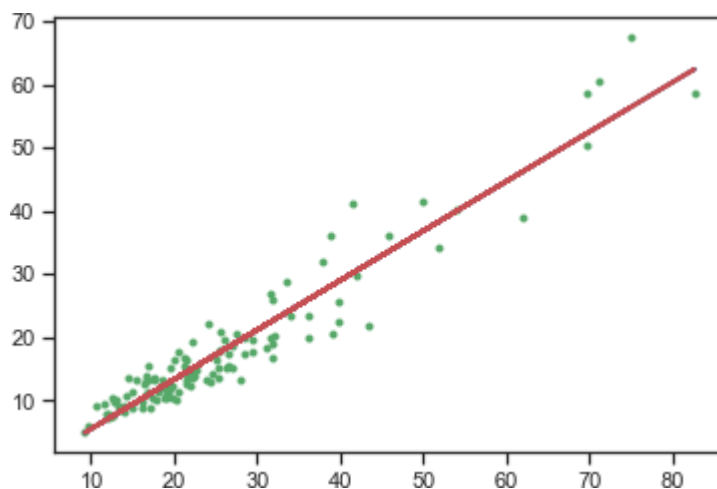
$b_0 = -2.3050406269840487$ - (теоретический), 0.03606307692307693 - (градиентный спуск)
 $b_1 = 0.7830951357632508$ - (теоретический), 1.2474327075213676 - (градиентный спуск)
 $MSE = 250.15077233076002$



Добьемся сходимости алгоритма путем задания других коэффициентов:

[33] : `xhgw_gradient_descent(100, -2.98, -22)`

$b_0 = -2.3050406269840487$ - (теоретический), -2.3021578094008817 - (градиентный спуск)
 $b_1 = 0.7830951357632508$ - (теоретический), 0.7830093489274242 - (градиентный спуск)
 $MSE = 1.889852950340935e-06$



Подбор коэффициентов через LinearRegression

```
[34] : from sklearn.linear_model import LinearRegression
```

Коэффициенты, полученные с использованием LinearRegression:

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

```
[35] : ((0.7830951357632508, array([[0.78309514]])),  
       (-2.3050406269840487, array([-2.30504063])))
```

Линейная модель:

```
[36] : model1 = LinearRegression()  
model1.fit(X_train, y_train)
```

```
[36] : LinearRegression()
```

Стохастический градиентный спуск с использованием SGDRegressor

```
[37] : from sklearn.linear_model import SGDRegressor
```

Коэффициенты, найденные с использованием SGDRegressor:

```
[38] : regr2 = SGDRegressor().fit(x_array.reshape(-1, 1), y_array)  
(b1, regr2.coef_), (b0, regr2.intercept_)
```

```
[38] : ((0.7830951357632508, array([-5765028.75808579])),  
       (-2.3050406269840487, array([-29211796.78292314])))
```

Линейная модель, построенная в этом случае:

```
[39] : model2 = SGDRegressor()  
model2.fit(X_train, y_train)
```

```
[39] : SGDRegressor()
```

L1-регуляризация линейной модели

```
[40] : from sklearn.linear_model import Lasso
```

Коэффициенты, полученные с использованием Lasso:

```
[41] : regr3 = Lasso().fit(x_array.reshape(-1, 1), y_array)  
(b1, regr3.coef_), (b0, regr3.intercept_)
```

```
[41] : ((0.7830951357632508, array([0.77805742])),  
       (-2.3050406269840487, -2.174213796155911))
```

Линейная модель:

```
[42] : model3 = Lasso()  
model3.fit(X_train, y_train)
```

```
[42] : Lasso()
```

2.3.2. SVM модель

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


```
from sklearn.svm import SVR
```

[43] :

Модель SVM:

```
regr4 = SVR(kernel='linear', C=1.0)
```

[44] : `model4 = regr4.fit(X_train, y_train)`

2.3.3. Дерево решений

[45] : `from sklearn.tree import DecisionTreeRegressor`

```
[46] : def xtat_tree(estimator):
    n_nodex = estimator.tree_.node_count
    children_left = estimator.tree_.children_left
    children_right = estimator.tree_.children_right

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

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

    print(DBCерого узлов:D, n_nodex)
    print(DЛистовых узлов:D, sum(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:

[47] : `regr5 = DecisionTreeRegressor(max_depth=3)`
`model5 = regr5.fit(X_train, y_train)`

Выведем основную статистику для дерева:

[48] : `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:

[49] : `regr6 = DecisionTreeRegressor(max_depth=5)`
`model6 = regr6.fit(X_train, y_train)`

```
[50] : 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.

2.4. 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):

```
[51] : from sklearn.metrics import mean_absolute_error, median_absolute_error
```

```
[52] : err1 = []  
      : err2 = []
```

Сформируем функцию для оценивания:

```
[53] : def rate_model(model):  
      :     print(DСредняя абсолютная ошибка:D,  
      :           mean_absolute_error(y_text, model.predict(X_text)))  
      :     err1.append(mean_absolute_error(y_text, model.predict(X_text)))  
      :     print(DМедианная абсолютная ошибка:D,  
      :           median_absolute_error(y_text, model.predict(X_text)))  
      :     err2.append(median_absolute_error(y_text, model.predict(X_text)))
```

```
[54] : rate_model(model1)
```

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

```
[55] : rate_model(model2)
```

Средняя абсолютная ошибка: 31007083086666.18
Медианная абсолютная ошибка: 28347016356507.68

```
[56] : rate_model(model3)
```

Средняя абсолютная ошибка: 0.11061266741068003
Медианная абсолютная ошибка: 0.08038352323471276

```
[57] : rate_model(model4)
```

Средняя абсолютная ошибка: 0.04773643904175788
Медианная абсолютная ошибка: 0.031832622636447816

```
[58] : rate_model(model5)
```

Средняя абсолютная ошибка: 1.4483803921568623
Медианная абсолютная ошибка: 1.372023172905526

```
[59] : rate_model(model6)
```

Средняя абсолютная ошибка: 0.7933329365079366

Медианная абсолютная ошибка: 0.332500000000000046

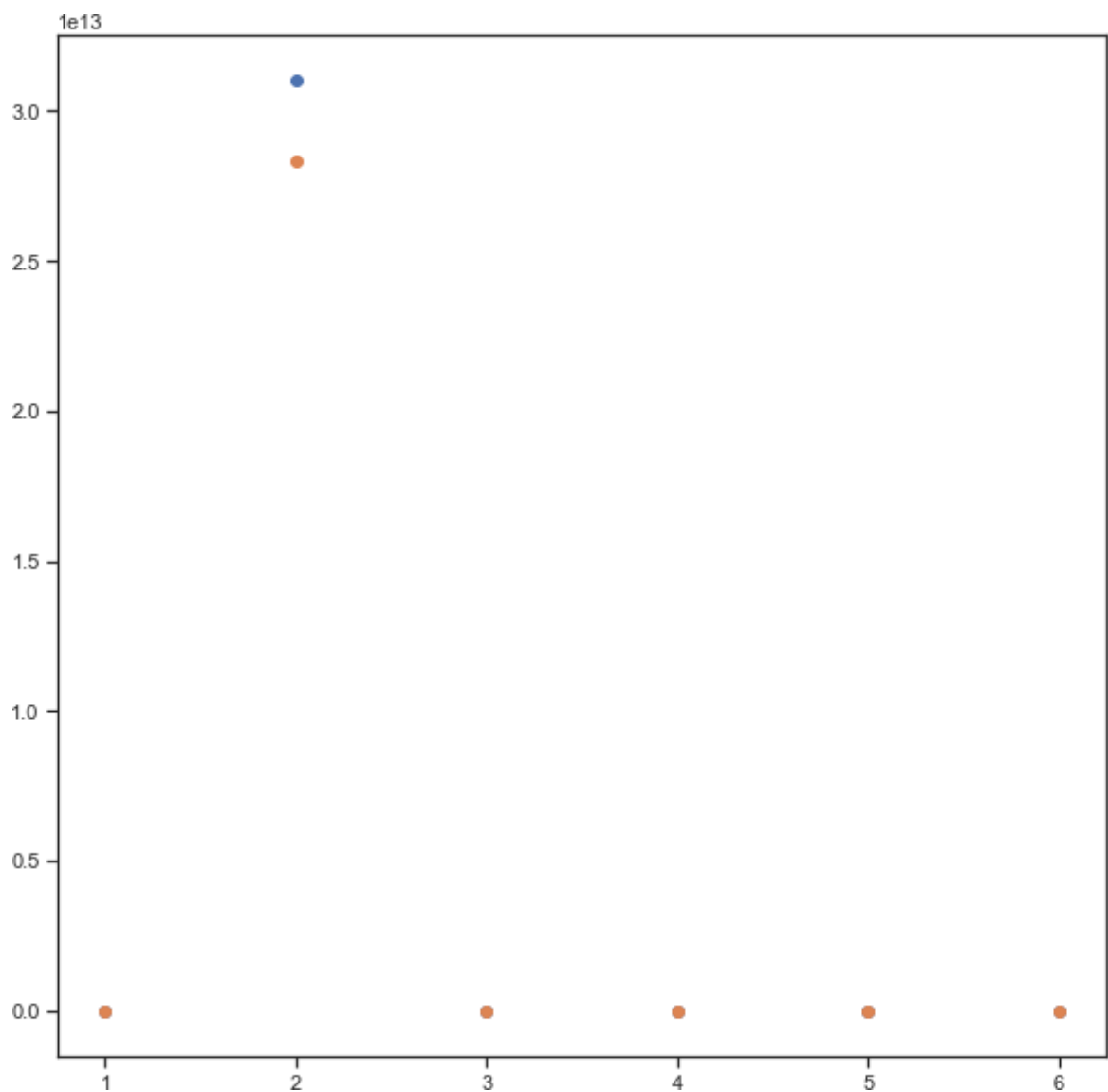
Чем ближе значение средней абсолютной ошибки и медианной абсолютной ошибки к нулю, тем лучше качество регрессии.

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

Визуальное представление оценки:

```
[60] : fig, ax = plt.subplots(figsize=(10,10))
      : plt.scatter([1, 2, 3, 4, 5, 6], err1)
      : plt.scatter([1, 2, 3, 4, 5, 6], err2)
```

```
[60] : <matplotlib.collections.PathCollection at 0x1f6c5084fa0>
```

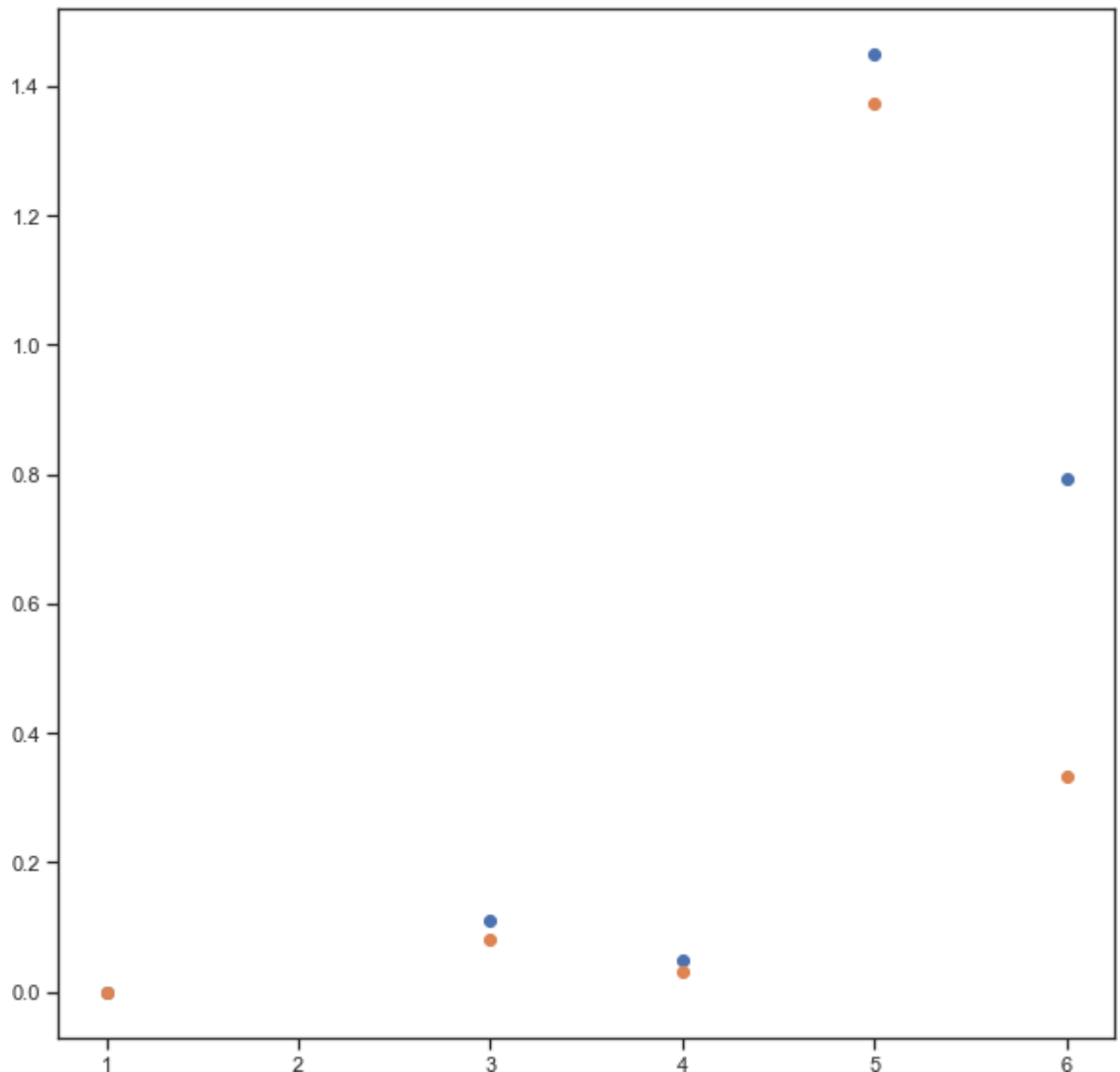


```
[61] : err1.pop(1)  
      : err2.pop(1)
```

```
[61]: 28347016356507.68
```

```
[62] : fig, ax = plt.subplots(figsize=(10,10))  
      : plt.scatter([1, 3, 4, 5, 6], err1)  
      : plt.scatter([1, 3, 4, 5, 6], err2)
```

```
[62] : <matplotlib.collections.PathCollection at 0x1f6c4fb76a0>
```



2.5. График важности признаков в дереве решений

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

```
[63] : from operator import itemgetter
```

```

def draw_feature_importancex(tree_model, X_datatset, figsize=(18,5)):
    """
    Вывод важности признаков в виде графика
    """
    # Сортировка значений важности признаков по убыванию
    list_to_xort = list(zip(X_datatset.columnx.valuex, tree_model.
feature_importancex_))
    xorted_list = xorted(list_to_xort, key=itemgetter(1), reverse = True)
    # Названия признаков
    labelx = [x for x, _ in xorted_list]
    # Важности признаков
    data = [x for _, x in xorted_list]
    # Вывод графика
    fig, ax = plt.subplots(figsize=figsize)
    ind = np.arange(len(labelx))
    plt.bar(ind, data)
    plt.xticks(ind, labelx, 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 labelx, data

```

Проверим для модели с глубиной = 3:

```
[64] : list(zip(data.columnx.valuex, model5.feature_importancex_))
```

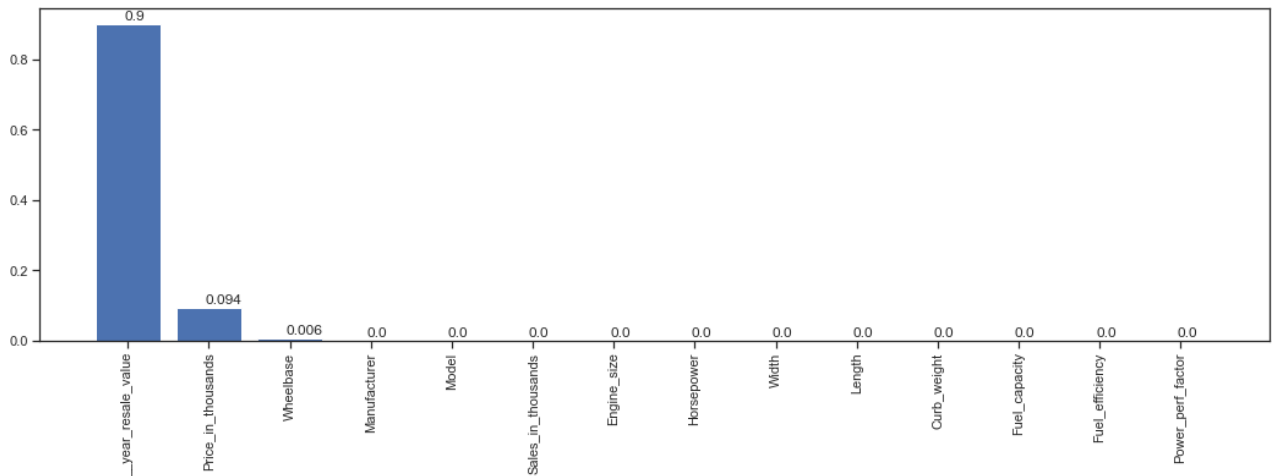
```

[64] : [('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),
        ('Horsepower', 0.0),
        ('Wheelbase', 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)]

```

```
[65] : car_tree_cl_fl_1, car_tree_cl_fd_1 = draw_feature_importancex(model5, data)
```



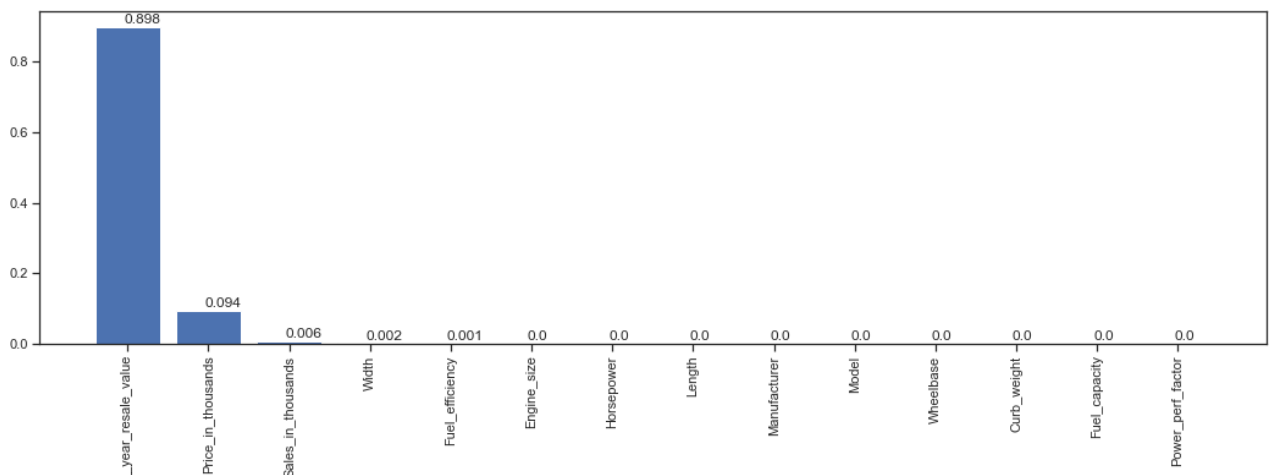
И для модели с глубиной = 5:

```
[66] : list(zip(data.columnx.valuex, model6.feature_importancex_))
```

```
[66] : [('Manufacturer', 0.0),
        ('Model', 0.0),
        ('Sales_in_thousands', 0.005686857877003804),
        ('__year_resale_value', 0.8976880644389448),

        ('Price_in_thousands', 0.09396949590123833),
        ('Engine_size', 0.00020496267968645642),
        ('Horsepower', 0.00020093402747307438),
        ('Wheelbase', 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)]
```

```
[67] : car_tree_cl_fl_2, car_tree_cl_fd_ = draw_feature_importancex(model6, data)
```



2.6. Визуализация деревьев решений и его правила

```
[68] : from io import StringIO
from IPython.display import Image
import graphviz
import pydotplus
from sklearn.tree import export_graphviz
```

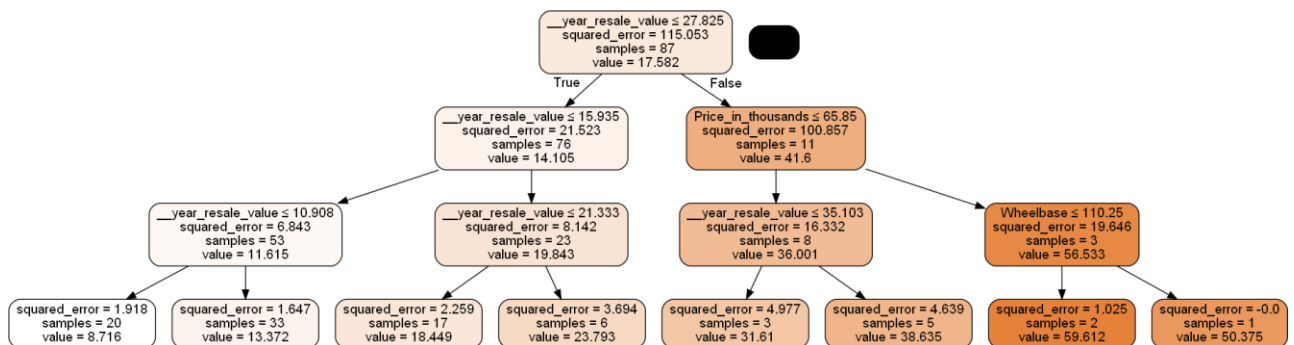
2.6.1. Визуализация деревьев

```
[69] : def get_png_tree(tree_model_param, feature_names_param):
    dot_data = StringIO()
    export_graphviz(tree_model_param, out_file=dot_data,
    feature_names=feature_names_param,
    filled=True, rounded=True, special_characters=True)
    graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
    return graph.create_png()
```

Визуализируем дерево с глубиной = 3:

```
[70] : Image(get_png_tree(model5, data.columnx), height='100%')
```

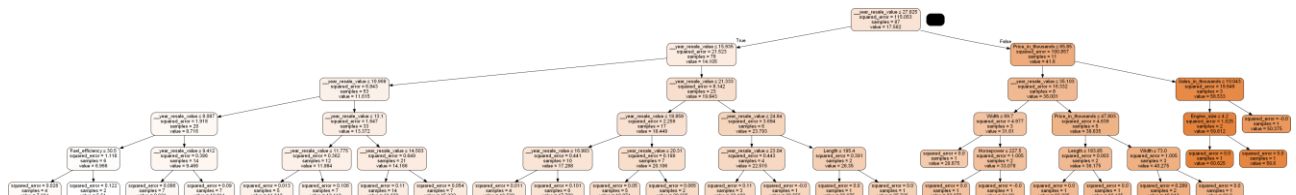
[70]:



И с глубиной = 5:

```
[71] : Image(get_png_tree(model6, data.columnx), height='100%')
```

[71]:



2.6.2. 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]

```