Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатика и система управления

КАФЕДРА Системы обработки информации и управления (ИУ5)

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

НА ТЕМУ:

Прогнозирование цен на автомобили

Студент ИУ5Ц-82Б (Группа) _____ (Подпись, дата) А.Н. Свинцов (И.О.Фамилия)

Руководитель _____ (Подпись, дата) Ю.Е.Гапанюк (И.О.Фамилия)

Консультант _____ (Подпись, дата) _____ (И.О.Фамилия)

2023 г.

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ

Заведующий кафедрой _____
(Индекс)

_____ _____
(И.О.Фамилия)

«_____»_____2023___г.

З А Д А Н И Е
на выполнение научно-исследовательской работы

по теме Прогнозирование цен на автомобили

Студент группы ИУ5Ц-82Б

Свинцов Артемий Николаевич
(Фамилия, имя, отчество)

Направленность НИР (учебная, исследовательская, практическая, производственная, др.)

Источник тематики (кафедра, предприятие, НИР) кафедра

График выполнения НИР: 25% к 3 нед., 50% к 9 нед., 75% к 12 нед., 100% к 15 нед.

Техническое задание *Решить задачу регрессии по прогнозированию цен на автомобили с использованием материалов дисциплины «Технологии машинного обучения»*

Оформление научно-исследовательской работы:

Расчетно-пояснительная записка на 37 листах формата А4.
Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)

Дата выдачи задания «22» февраля 2023 г.

Руководитель НИР _____ Ю.Е.Гапанюк
(Подпись, дата) (И.О.Фамилия)

Студент _____ А.Н. Свинцов
(Подпись, дата) (И.О.Фамилия)

Примечание: Задание оформляется в двух экземплярах: один выдается студенту, второй хранится на кафедре.

СОДЕРЖАНИЕ

1. Введение

В качестве предметной области был выбран набор данных, содержащий данные об автомобилях, проданных за некоторый период на территории США.

Задача данной работы - предсказание цены автомобиля на основе нескольких факторов. Данная задача может быть актуальна для автомобильной компании, планирующей свой выход на автомобильный рынок США, открыв там свое производственное предприятие и производя автомобили локально, чтобы составить конкуренцию своим американским и европейским аналогам.

Решение этой задачи может быть использовано руководством автомобильной компании для понимания того, как именно цены изменяются в зависимости от характеристик автомобилей. С использованием этих данных, оно сможет более оптимально разарабатывать новые модели своих автомобилей, чтобы соответствовать определенным ценовым сегментам. Кроме того, построенная модель регрессии может стать хорошим способом для понимания динамики ценообразования на новом рынке.

2. Основная часть

2.1. Поставновка задачи

Необходимо решить задачу регрессии по прогнозированию цен на автомобили с использованием материалов дисциплины «Технологии машинного обучения».

2.2. Выбор набора данных для построения моделей машинного обучения

2.2.1. Текстовое описание

Данный набор доступен по адресу: https://www.kaggle.com/datasets/goyalshalini93/car-data
Набор данных имеет следующие атрибуты:

- car_ID - порядковый номер строки
- symboling - обозначение
- CarName - марка + модель автомобиля
- fueltype - тип топлива
- aspiration - тип подачи воздуха в двигатель (атмосферный/турбированный)
- doornumber - число дверей
- carbody - тип кузова
- drivewheel - привод
- enginelocation - расположение двигателя
- wheelbase - длина колесной базы
- carlength - длина автомобиля
- carwidth - ширина автомобиля
- carheight - высота автомобиля
- curbweight - снаряженная масса
- enginetype - тип двигателя
- cylindernumber - число цилиндров
- enginesize - объем двигателя
- fuelsystem - тип топливной системы
- boreratio - интерес для покупателя
- stroke - поршни
- compressionratio - компрессия
- horsepower - лошадиные силы
- peakrpm - обороты в минуты, при которых достигается максимальный момент
- citympg - расход топлива по городу
- highwaympg - расход по трассе
- price - цена

Решается задача регрессии. В качестве целевого признака - цена.

2.2.2. Импорт библиотек

Импортируем необходимые начальные библиотеки:

[1]:
```
import warnings
warnings.filterwarnings('ignore')

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

2.2.3. Загрузка данных

Загрузим данные:

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

2.3. Разведочный анализ данных

2.3.1. Основные характеристики

Первые 5 строк датасета:

[3]:
```
data.head()
```

[3]:

	car_ID	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	wheelbase	⋯	enginesize	fuelsystem	boreratio	stroke	compressionratio	hor	sepower	peakrpm	citympg	highwaympg	price
0	1	3	alfa-romero giulia	gas	std	two	convertible	rwd	front	88.6	⋯	130	mpfi	3.47	2.68	9.0		111	5000	21	27	13495.0
1	2	3	alfa-romero stelvio	gas	std	two	convertible	rwd	front	88.6	⋯	130	mpfi	3.47	2.68	9.0		111	5000	21	27	16500.0
2	3	1	alfa-romero Quadrifoglio	gas	std	two	hatchback	rwd	front	94.5	⋯	152	mpfi	2.68	3.47	9.0		154	5000	19	26	16500.0
3	4	2	audi 100 ls	gas	std	four	sedan	fwd	front	99.8	⋯	109	mpfi	3.19	3.40	10.0		102	5500	24	30	13950.0
4	5	2	audi 100ls	gas	std	four	sedan	4wd	front	99.4	⋯	136	mpfi	3.19	3.40	8.0		115	5500	18	22	17450.0

[5 rows x 26 columns]

Размер датасета:

[4]:
```
data.shape
```

[4]:
```
(205, 26)
```

Столбцы:

[5]:
```
data.columns
```

[5]:
```
Index(['car_ID', 'symboling', 'CarName', 'fueltype', 'aspiration',
       'doornumber', 'carbody', 'drivewheel', 'enginelocation', 'wheelbase',
       'carlength', 'carwidth', 'carheight', 'curbweight', 'enginetype',
       'cylindernumber', 'enginesize', 'fuelsystem', 'boreratio', 'stroke',
       'compressionratio', 'horsepower', 'peakrpm', 'citympg', 'highwaympg',
       'price'],
      dtype='object')
```

Типы данных:

[6]:
```
data.dtypes
```

[6]:
```
car_ID              int64
symboling           int64
CarName            object
fueltype           object
aspiration         object
doornumber         object
carbody            object
drivewheel         object
enginelocation     object
wheelbase         float64
carlength         float64
carwidth          float64
carheight         float64
curbweight          int64
enginetype         object
cylindernumber     object
enginesize          int64
fuelsystem         object
boreratio         float64
stroke            float64
compressionratio  float64
horsepower          int64
peakrpm             int64
citympg             int64
highwaympg          int64
price             float64
dtype: object
```

2.3.2. Обработка данных с неинформативными признаками

В датасете присутствуют данные, которые не несут полезной информации для дальнейшего анализа.

Аналитически посчитаем неинформативные признаки (у которых более 90% строк имеют одинаковое значение):

[7]:
```
num_rows = len(data.index)
low_information_cols = [] #

for col in data.columns:
    cnts = data[col].value_counts(dropna=False)
    top_pct = (cnts/num_rows).iloc[0]

    if top_pct > 0.90:
        low_information_cols.append(col)
        print('{0}: {1:.5f}%'.format(col, top_pct*100))
        print(cnts)
        print()
```

```
fueltype: 90.24390%
gas       185
diesel     20
Name: fueltype, dtype: int64

enginelocation: 98.53659%
front    202
rear       3
Name: enginelocation, dtype: int64
```

Удалим соответствующие столбцы:

[8]:
```
data.drop(['fueltype', 'enginelocation'], inplace=True, axis=1)
```

Некоторые столбцы также не представляют ценности для дальнейшего анализа. Также удалим их:

[9]:
```
data.drop(['car_ID', 'symboling', 'enginesize', 'stroke', 'compressionratio'],
↪inplace=True, axis=1)
```

Проверим корректность удаления:

[10]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 19 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   CarName         205 non-null    object
 1   aspiration      205 non-null    object
 2   doornumber      205 non-null    object
 3   carbody         205 non-null    object
 4   drivewheel      205 non-null    object
 5   wheelbase       205 non-null    float64
 6   carlength       205 non-null    float64
 7   carwidth        205 non-null    float64
 8   carheight       205 non-null    float64
 9   curbweight      205 non-null    int64
 10  enginetype      205 non-null    object
 11  cylindernumber  205 non-null    object
 12  fuelsystem      205 non-null    object
```

```
 13  boreratio       205 non-null    float64
 14  horsepower      205 non-null    int64
 15  peakrpm         205 non-null    int64
 16  citympg         205 non-null    int64
 17  highwaympg      205 non-null    int64
 18  price           205 non-null    float64
dtypes: float64(6), int64(5), object(8)
memory usage: 30.6+ KB
```

2.3.3. Обработка пропусков

Определим столбцы с пропусками данных:

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

[11]:
```
CarName           0
aspiration        0
doornumber        0
carbody           0
drivewheel        0
wheelbase         0
carlength         0
carwidth          0
carheight         0
curbweight        0
enginetype        0
cylindernumber    0
fuelsystem        0
boreratio         0
horsepower        0
peakrpm           0
citympg           0
highwaympg        0
price             0
dtype: int64
```

Видим, что в наборе данных отсутствуют пропуски.

2.3.4. Переименование столбцов

Для более удобной дальнейшей работы переименуем столбцы:

[12]:
```
data.rename(columns = {'doornumber' : 'doors', 'carbody' : 'body', 'drivewheel' :
↪'drive', 'carlength' : 'length', 'carwidth' : 'width', 'carheight' : 'height',
↪'curbweight' : 'weight', 'cylindernumber' : 'cyl', 'boreratio' : 'bore'},
↪inplace = True)
```

[13]:
```
data.head()
```

[13]:

	CarName	aspiration	doors	body	drive	wheelbase	\
0	alfa-romero giulia	std	two	convertible	rwd	88.6	
1	alfa-romero stelvio	std	two	convertible	rwd	88.6	
2	alfa-romero Quadrifoglio	std	two	hatchback	rwd	94.5	
3	audi 100 ls	std	four	sedan	fwd	99.8	
4	audi 100ls	std	four	sedan	4wd	99.4	

	length	width	height	weight	enginetype	cyl	fuelsystem	bore	\
0	168.8	64.1	48.8	2548	dohc	four	mpfi	3.47	
1	168.8	64.1	48.8	2548	dohc	four	mpfi	3.47	
2	171.2	65.5	52.4	2823	ohcv	six	mpfi	2.68	
3	176.6	66.2	54.3	2337	ohc	four	mpfi	3.19	
4	176.6	66.4	54.3	2824	ohc	five	mpfi	3.19	

	horsepower	peakrpm	citympg	highwaympg	price
0	111	5000	21	27	13495.0
1	111	5000	21	27	16500.0
2	154	5000	19	26	16500.0
3	102	5500	24	30	13950.0
4	115	5500	18	22	17450.0

2.3.5. Преобразование столбцов

Преобразуем столбец, содержащий информацию о марке и модели, к двум отдельным столбцам:

[14]:
```
data[['manuf', 'model']] = data['CarName'].str.split(' ', 1, expand=True)
data.drop(['CarName'],axis=1,inplace=True)
data = data[['manuf', 'model', 'aspiration', 'doors', 'body', 'drive',
 ↪'wheelbase', 'length', 'width', 'height', 'weight', 'enginetype', 'cyl',
 ↪'fuelsystem', 'bore', 'horsepower', 'peakrpm', 'citympg', 'highwaympg',
 ↪'price']]
data.head()
```

[14]:

	manuf	model	aspiration	doors	body	drive	wheelbase	\
0	alfa-romero	giulia	std	two	convertible	rwd	88.6	
1	alfa-romero	stelvio	std	two	convertible	rwd	88.6	
2	alfa-romero	Quadrifoglio	std	two	hatchback	rwd	94.5	
3	audi	100 ls	std	four	sedan	fwd	99.8	
4	audi	100ls	std	four	sedan	4wd	99.4	

	length	width	height	weight	enginetype	cyl	fuelsystem	bore	\
0	168.8	64.1	48.8	2548	dohc	four	mpfi	3.47	
1	168.8	64.1	48.8	2548	dohc	four	mpfi	3.47	
2	171.2	65.5	52.4	2823	ohcv	six	mpfi	2.68	
3	176.6	66.2	54.3	2337	ohc	four	mpfi	3.19	
4	176.6	66.4	54.3	2824	ohc	five	mpfi	3.19	

	horsepower	peakrpm	citympg	highwaympg	price
0	111	5000	21	27	13495.0
1	111	5000	21	27	16500.0
2	154	5000	19	26	16500.0
3	102	5500	24	30	13950.0
4	115	5500	18	22	17450.0

2.3.6. Исправление ошибок

Проверим наличие ошибок:

[15]:
```
data.manuf.unique()
```

[15]:
```
array(['alfa-romero', 'audi', 'bmw', 'chevrolet', 'dodge', 'honda',
       'isuzu', 'jaguar', 'maxda', 'mazda', 'buick', 'mercury',
       'mitsubishi', 'Nissan', 'nissan', 'peugeot', 'plymouth', 'porsche',
       'porcshce', 'renault', 'saab', 'subaru', 'toyota', 'toyouta',
       'vokswagen', 'volkswagen', 'vw', 'volvo'], dtype=object)
```

[16]:
```
data.model.unique()
```

[16]:
```
array(['giulia', 'stelvio', 'Quadrifoglio', '100 ls', '100ls', 'fox',
       '5000', '4000', '5000s (diesel)', '320i', 'x1', 'x3', 'z4', 'x4',
       'x5', 'impala', 'monte carlo', 'vega 2300', 'rampage',
       'challenger se', 'd200', 'monaco (sw)', 'colt hardtop',
       'colt (sw)', 'coronet custom', 'dart custom',
       'coronet custom (sw)', 'civic', 'civic cvcc', 'accord cvcc',
       'accord lx', 'civic 1500 gl', 'accord', 'civic 1300', 'prelude',
       'civic (auto)', 'MU-X', 'D-Max ', 'D-Max V-Cross', 'xj', 'xf',
       'xk', 'rx3', 'glc deluxe', 'rx2 coupe', 'rx-4', '626', 'glc',
       'rx-7 gs', 'glc 4', 'glc custom l', 'glc custom',
       'electra 225 custom', 'century luxus (sw)', 'century', 'skyhawk',
       'opel isuzu deluxe', 'skylark', 'century special',
       'regal sport coupe (turbo)', 'cougar', 'mirage', 'lancer',
       'outlander', 'g4', 'mirage g4', 'montero', 'pajero', 'versa',
       'gt-r', 'rogue', 'latio', 'titan', 'leaf', 'juke', 'note',
       'clipper', 'nv200', 'dayz', 'fuga', 'otti', 'teana', 'kicks',
       '504', '304', '504 (sw)', '604sl', '505s turbo diesel', 'fury iii',
       'cricket', 'satellite custom (sw)', 'fury gran sedan', 'valiant',
       'duster', 'macan', 'panamera', 'cayenne', 'boxter', '12tl',
       '5 gtl', '99e', '99le', '99gle', None, 'dl', 'brz', 'baja', 'r1',
       'r2', 'trezia', 'tribeca', 'corona mark ii', 'corona',
       'corolla 1200', 'corona hardtop', 'corolla 1600 (sw)', 'carina',
       'mark ii', 'corolla', 'corolla liftback', 'celica gt liftback',
       'corolla tercel', 'corona liftback', 'starlet', 'tercel',
       'cressida', 'celica gt', 'rabbit', '1131 deluxe sedan',
       'model 111', 'type 3', '411 (sw)', 'super beetle', 'dasher',
       'rabbit custom', '145e (sw)', '144ea', '244dl', '245', '264gl',
       'diesel', '246'], dtype=object)
```

[17]:
```
data.aspiration.unique()
```

[17]:
```
array(['std', 'turbo'], dtype=object)
```

[18]:
```
data.doors.unique()
```

[18]:
```
array(['two', 'four'], dtype=object)
```

[19]:
```
data.body.unique()
```

[19]:
```
array(['convertible', 'hatchback', 'sedan', 'wagon', 'hardtop'],
      dtype=object)
```

[20]:
```
data.drive.unique()
```

[20]:
```
array(['rwd', 'fwd', '4wd'], dtype=object)
```

[21]:
```
data.enginetype.unique()
```

[21]:
```
array(['dohc', 'ohcv', 'ohc', 'l', 'rotor', 'ohcf', 'dohcv'], dtype=object)
```

[22]:
```
data.cyl.unique()
```

[22]:
```
array(['four', 'six', 'five', 'three', 'twelve', 'two', 'eight'],
      dtype=object)
```

[23]:
```
data.fuelsystem.unique()
```

[23]:
```
array(['mpfi', '2bbl', 'mfi', '1bbl', 'spfi', '4bbl', 'idi', 'spdi'],
      dtype=object)
```

В столбце производителя автомобилей есть небольшие ошибки. Исправим их:

[24]:
```
data.manuf = data.manuf.str.lower()

def replace_name(a,b):
    data.manuf.replace(a,b,inplace=True)

replace_name('maxda','mazda')
replace_name('porcshce','porsche')
replace_name('toyouta','toyota')
replace_name('vokswagen','volkswagen')
replace_name('vw','volkswagen')

data.manuf.unique()
```

[24]:
```
array(['alfa-romero', 'audi', 'bmw', 'chevrolet', 'dodge', 'honda',
       'isuzu', 'jaguar', 'mazda', 'buick', 'mercury', 'mitsubishi',
       'nissan', 'peugeot', 'plymouth', 'porsche', 'renault', 'saab',
       'subaru', 'toyota', 'volkswagen', 'volvo'], dtype=object)
```

[25]:
```
data.head()
```

[25]:

	manuf	model	aspiration	doors	body	drive	wheelbase	\
0	alfa-romero	giulia	std	two	convertible	rwd	88.6	
1	alfa-romero	stelvio	std	two	convertible	rwd	88.6	
2	alfa-romero	Quadrifoglio	std	two	hatchback	rwd	94.5	
3	audi	100 ls	std	four	sedan	fwd	99.8	
4	audi	100ls	std	four	sedan	4wd	99.4	

	length	width	height	weight	enginetype	cyl	fuelsystem	bore	\
0	168.8	64.1	48.8	2548	dohc	four	mpfi	3.47	
1	168.8	64.1	48.8	2548	dohc	four	mpfi	3.47	
2	171.2	65.5	52.4	2823	ohcv	six	mpfi	2.68	
3	176.6	66.2	54.3	2337	ohc	four	mpfi	3.19	
4	176.6	66.4	54.3	2824	ohc	five	mpfi	3.19	

	horsepower	peakrpm	citympg	highwaympg	price
0	111	5000	21	27	13495.0
1	111	5000	21	27	16500.0
2	154	5000	19	26	16500.0
3	102	5500	24	30	13950.0
4	115	5500	18	22	17450.0

2.3.7. Замена данных

В столбцах “doors” и “cyl” - объекты типа Object, числовые данные записаны в виде набора символов. Преобразуем их в числа:

[26]:
```
doors = {'two': 2, 'four': 4}
data['doors'] = data['doors'].replace(doors)
data['doors'] = data['doors'].astype({DdoorsD:Dint64D})

cyl = {'four': 4, 'six': 6, 'five': 5, 'three': 3, 'twelve': 12, 'two': 2,
 ↪'eight': 8}
data['cyl'] = data['cyl'].replace(cyl)
data['cyl'] = data['cyl'].astype({DcylD:Dint64D})
data.head()
```

[26]:

	manuf	model	aspiration	doors	body	drive	wheelbase	\
0	alfa-romero	giulia	std	2	convertible	rwd	88.6	
1	alfa-romero	stelvio	std	2	convertible	rwd	88.6	
2	alfa-romero	Quadrifoglio	std	2	hatchback	rwd	94.5	
3	audi	100 ls	std	4	sedan	fwd	99.8	
4	audi	100ls	std	4	sedan	4wd	99.4	

	length	width	height	weight	enginetype	cyl	fuelsystem	bore	horsepower	\
0	168.8	64.1	48.8	2548	dohc	4	mpfi	3.47	111	
1	168.8	64.1	48.8	2548	dohc	4	mpfi	3.47	111	
2	171.2	65.5	52.4	2823	ohcv	6	mpfi	2.68	154	
3	176.6	66.2	54.3	2337	ohc	4	mpfi	3.19	102	
4	176.6	66.4	54.3	2824	ohc	5	mpfi	3.19	115	

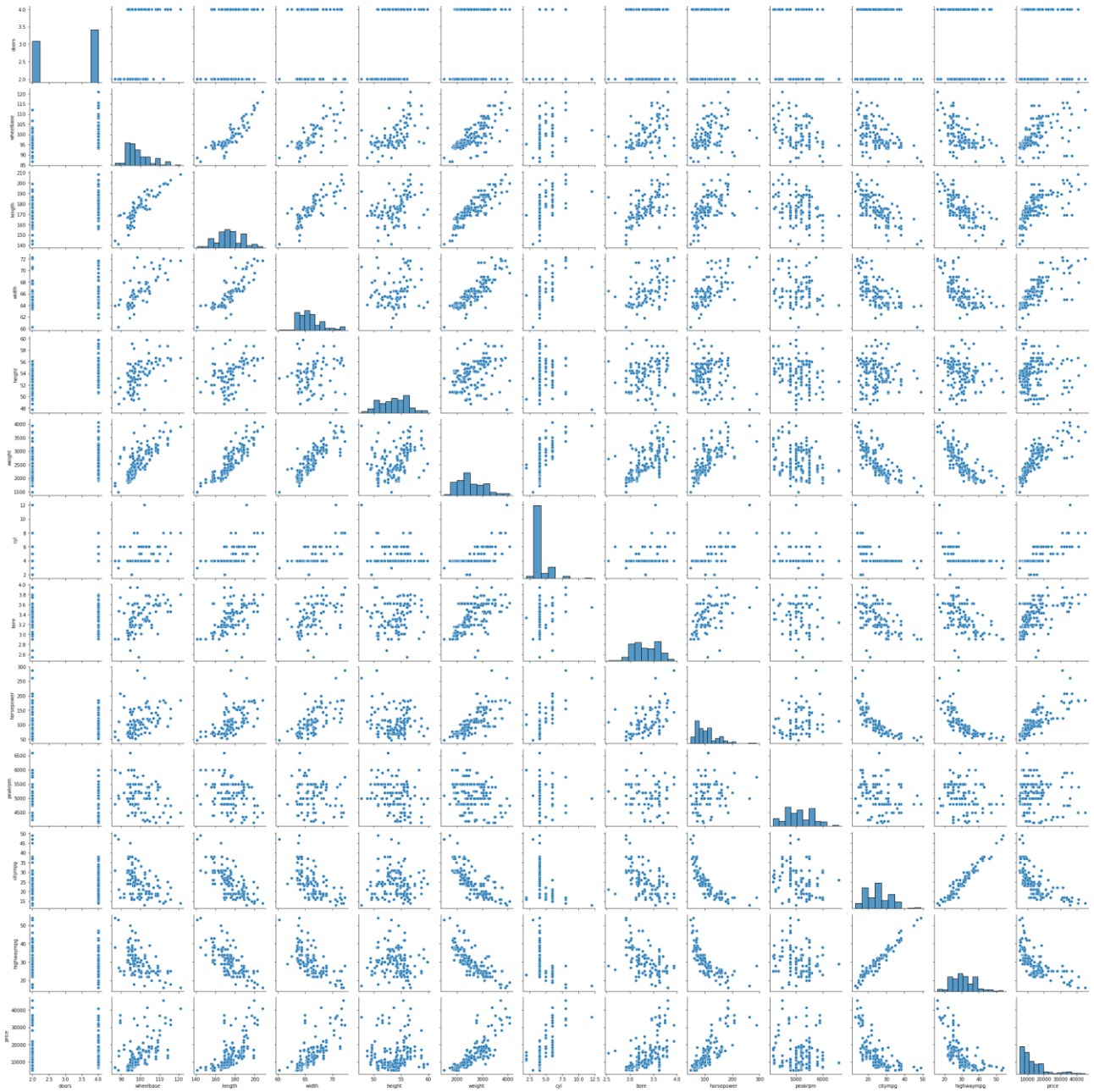
	peakrpm	citympg	highwaympg	price
0	5000	21	27	13495.0
1	5000	21	27	16500.0
2	5000	19	26	16500.0
3	5500	24	30	13950.0
4	5500	18	22	17450.0

2.3.8. Структура данных

Построим множество графиков, отображающих структуру данных:

[27]:
```
sns.pairplot(data)
```

[27]: <seaborn.axisgrid.PairGrid at 0x22a1cb5ba30>

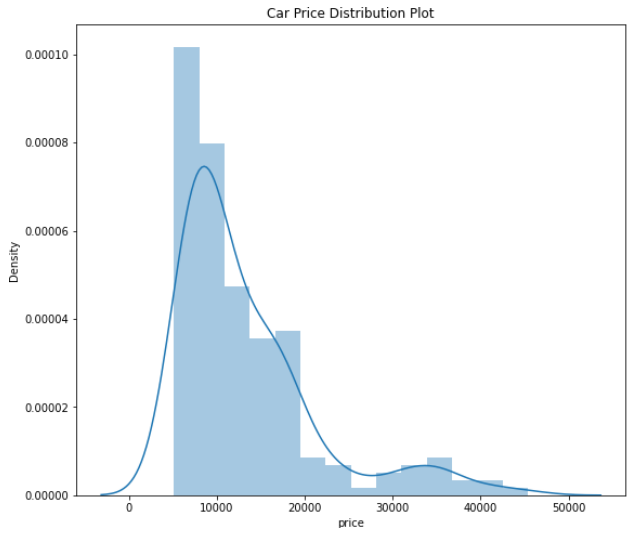

doors
wheelbase
length
width
height
weight
cyl
bore
horsepower
peakrpm
citympg
highwaympg
price

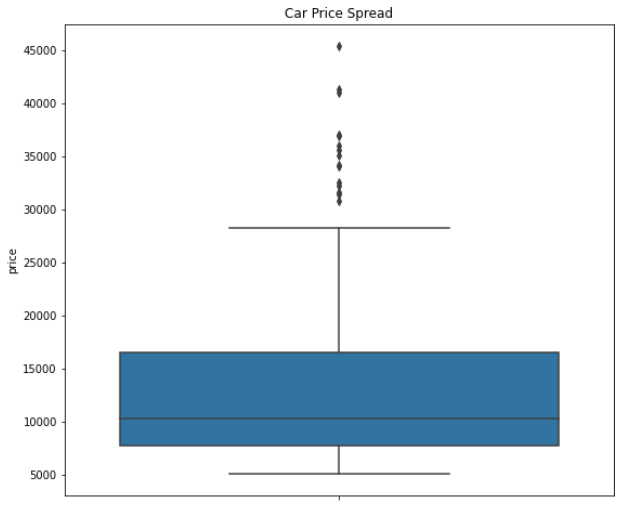
Построим графики относительно целевого признака price.

```
sns.pairplot(data, hue="price")
```

Построим графики распределения цен:

[28]:
```
plt.figure(figsize=(20,8))

plt.subplot(1,2,1)
plt.title('Car Price Distribution Plot')
sns.distplot(data.price)

plt.subplot(1,2,2)
plt.title('Car Price Spread')
sns.boxplot(y=data.price)
```





2.4. Кодирование категориальных признаков и масштабирование данных

Определим типы данных в наборе:

[29]:
```
data.dtypes
```

[29]:
```
manuf          object
model          object
aspiration     object
doors           int64
body           object
drive          object
wheelbase     float64
length        float64
width         float64
height        float64
weight          int64
enginetype     object
cyl             int64
fuelsystem     object
bore          float64
horsepower      int64
peakrpm         int64
citympg         int64
highwaympg      int64
price         float64
dtype: object
```

2.4.1. Кодирование категориальных признаков

Используя LabelEncoder из sckit-learn закодируем некоторые столбцы типа Object в числовые значения:

[30]:
```
from sklearn.preprocessing import LabelEncoder
```

[31]:
```
letypemanuf = LabelEncoder()
learrmanuf = letypemanuf.fit_transform(data[DmanufD])
data[DmanufD] = learrmanuf
data = data.astype({DmanufD:Dint64D})
```

[32]:
```
letypemodel = LabelEncoder()
learrmodel = letypemodel.fit_transform(data[DmodelD])
data[DmodelD] = learrmodel
data = data.astype({DmodelD:Dint64D})
```

[33]:
```
letypeasp = LabelEncoder()
learrasp = letypeasp.fit_transform(data[DaspirationD])
data[DaspirationD] = learrasp
data = data.astype({DaspirationD:Dint64D})
```

[34]:
```
letypebody = LabelEncoder()
learrbody = letypebody.fit_transform(data[DbodyD])
data[DbodyD] = learrbody
data = data.astype({DbodyD:Dint64D})
```

[35]:
```
letypedrive = LabelEncoder()
learrdrive = letypedrive.fit_transform(data[DdriveD])
data[DdriveD] = learrdrive
data = data.astype({DdriveD:Dint64D})
```

[36]:
```
letypetype = LabelEncoder()
learrtype = letypetype.fit_transform(data[DenginetypeD])
data[DenginetypeD] = learrtype
data = data.astype({DenginetypeD:Dint64D})
```

[37]:
```
letypefs = LabelEncoder()
learrfs = letypefs.fit_transform(data[DfuelsystemD])
data[DfuelsystemD] = learrfs
data = data.astype({DfuelsystemD:Dint64D})
```

[38]:
```
data.head()
```

[38]:

	manuf	model	aspiration	doors	body	drive	whee base	length	width	\
0	0	78	0	2	0	2	88.6	168.8	64.1	
1	0	122	0	2	0	2	88.6	168.8	64.1	
2	0	28	0	2	2	2	94.5	171.2	65.5	
3	1	0	0	4	3	1	99.8	176.6	66.2	
4	1	1	0	4	3	0	99.4	176.6	66.4	

	height	weight	enginetype	cyl	fuelsystem	bore	horsepower	peakrpm	\
0	48.8	2548	0	4	5	3.47	111	5000	
1	48.8	2548	0	4	5	3.47	111	5000	
2	52.4	2823	5	6	5	2.68	154	5000	
3	54.3	2337	3	4	5	3.19	102	5500	
4	54.3	2824	3	5	5	3.19	115	5500	

	citympg	highwaympg	price
0	21	27	13495.0
1	21	27	16500.0

```
2        19           26   16500.0
3        24           30   13950.0
4        18           22   17450.0
```

2.4.2. Масштабирование данных

Проведем масштабирование данных MinMax с помощью средств из sckit-learn:

[39]:
```
from sklearn.preprocessing import MinMaxScaler
```

[40]:
```
scaler = MinMaxScaler()
scaler_data = scaler.fit_transform(data[data.columns])
```

Сохраним масштабированные данные:

[41]:
```
data_scaled = pd.DataFrame()
```

[42]:
```
for i in range(len(data.columns)):
    col = data.columns[i]
    new_col_name = col + '_scaled'
    data_scaled[new_col_name] = scaler_data[:,i]
```

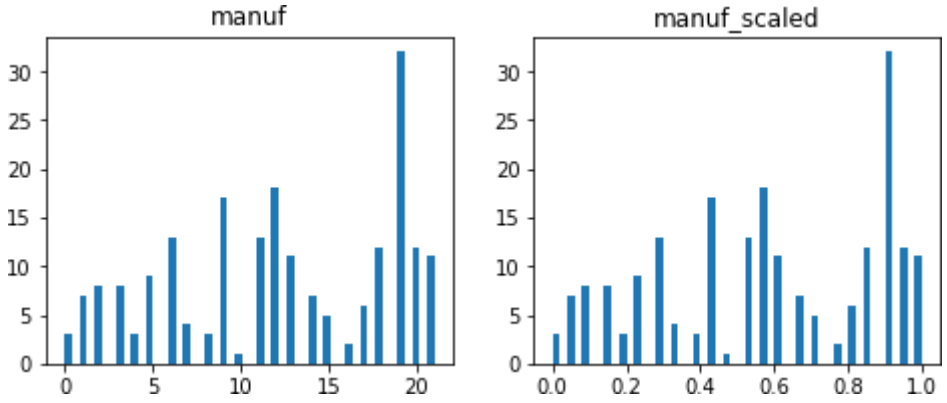
[43]:
```
data_scaled.head()
```

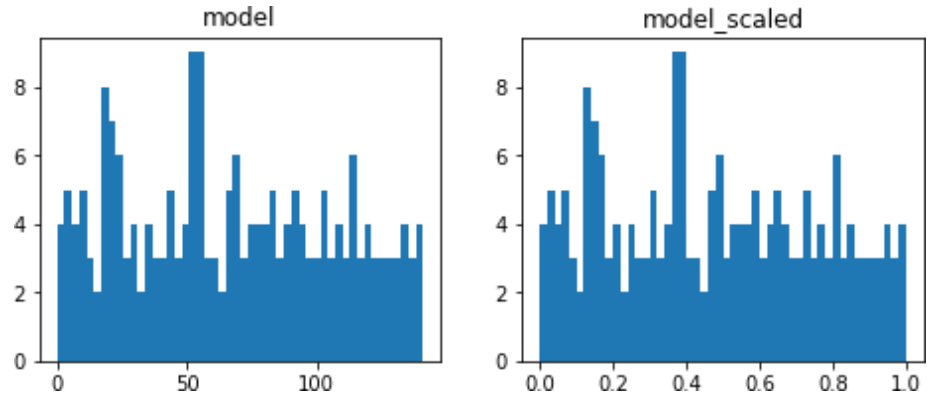
[43]:
```
   manuf_scaled  model_scaled  aspiration_scaled  doors_scaled  body_scaled  \
0      0.000000      0.553191                0.0           0.0         0.00
1      0.000000      0.865248                0.0           0.0         0.00
2      0.000000      0.198582                0.0           0.0         0.50
3      0.047619      0.000000                0.0           1.0         0.75
4      0.047619      0.007092                0.0           1.0         0.75

   drive_scaled  wheelbase_scaled  length_scaled  width_scaled  height_scaled  \
0           1.0          0.058309       0.413433      0.316667       0.083333
1           1.0          0.058309       0.413433      0.316667       0.083333
2           1.0          0.230321       0.449254      0.433333       0.383333
3           0.5          0.384840       0.529851      0.491667       0.541667
4           0.0          0.373178       0.529851      0.508333       0.541667

   weight_scaled  enginetype_scaled  cyl_scaled  fuelsystem_scaled  \
0       0.411171           0.000000         0.2           0.714286
1       0.411171           0.000000         0.2           0.714286
2       0.517843           0.833333         0.4           0.714286
3       0.329325           0.500000         0.2           0.714286
4       0.518231           0.500000         0.3           0.714286

   bore_scaled  horsepower_scaled  peakrpm_scaled  citympg_scaled  \
0     0.664286           0.262500        0.346939        0.222222
1     0.664286           0.262500        0.346939        0.222222
2     0.100000           0.441667        0.346939        0.166667
3     0.464286           0.225000        0.551020        0.305556
4     0.464286           0.279167        0.551020        0.138889

   highwaympg_scaled  price_scaled
0           0.289474      0.207959
1           0.289474      0.282558
```

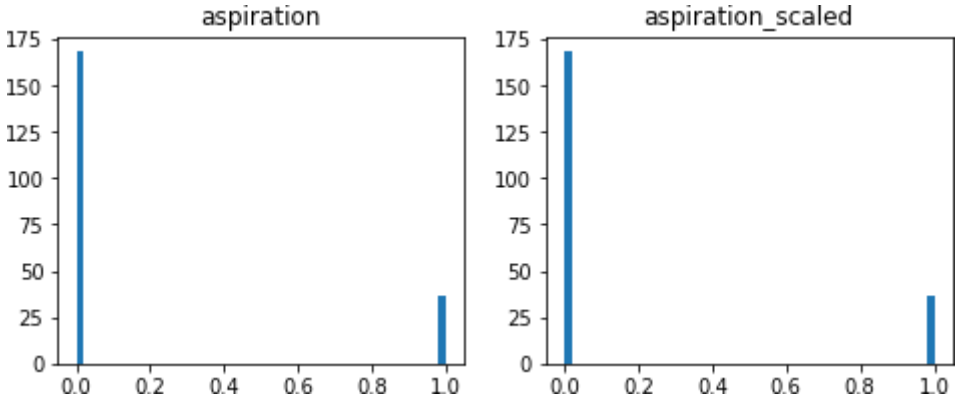
2	0.263158	0.282558
3	0.368421	0.219254
4	0.157895	0.306142

Масштабирование данных не повлияло на на распределение данных:

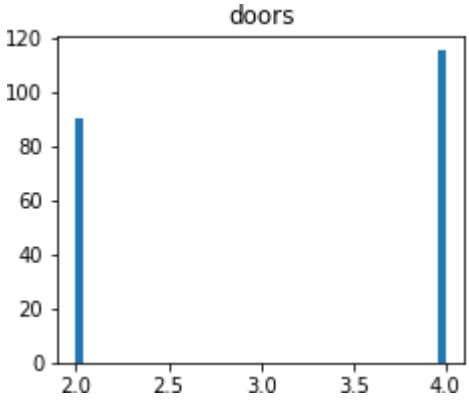
[44]:
```
for col in data.columns:
    col_scaled = col + '_scaled'

    fig, ax = plt.subplots(1, 2, figsize=(8,3))
    ax[0].hist(data[col], 50)
    ax[1].hist(data_scaled[col_scaled], 50)
    ax[0].title.set_text(col)
    ax[1].title.set_text(col_scaled)
    plt.show()
```

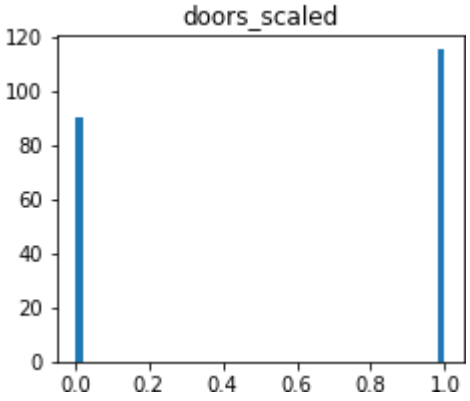


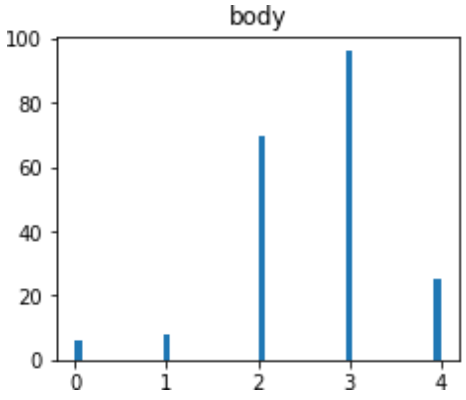


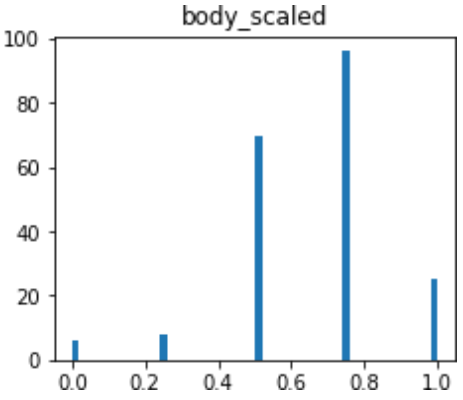

aspiration
175
150
125
100
75
50
25
0
0.0
0.2
0.4
0.6
0.8
1.0

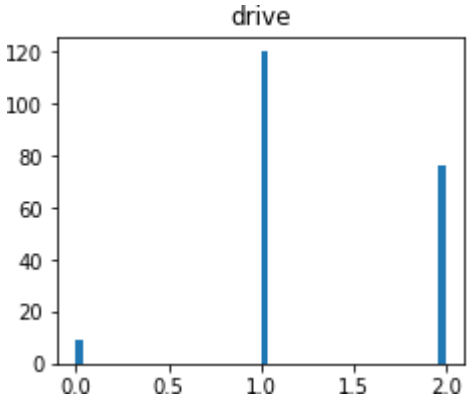
aspiration_scaled
175
150
125
100
75
50
25
0
0.0
0.2
0.4
0.6
0.8
1.0

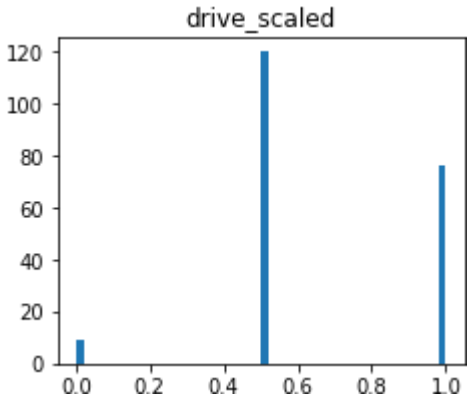

doors
120
100
80
60
40
20
0
2.0
2.5
3.0
3.5
4.0

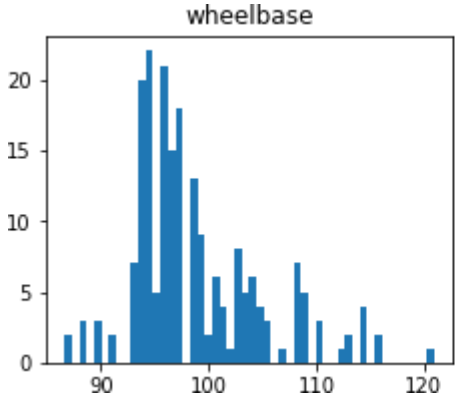

doors_scaled
120
100
80
60
40
20
0
0.0
0.2
0.4
0.6
0.8
1.0

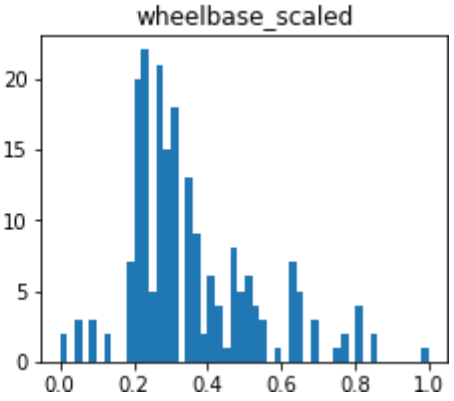

body
100
80
60
40
20
0
0
1
2
3
4

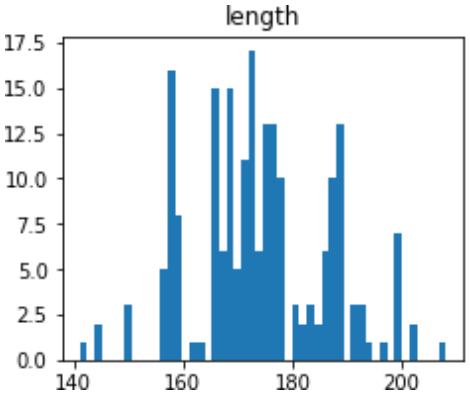

body_scaled
100
80
60
40
20
0
0.0
0.2
0.4
0.6
0.8
1.0

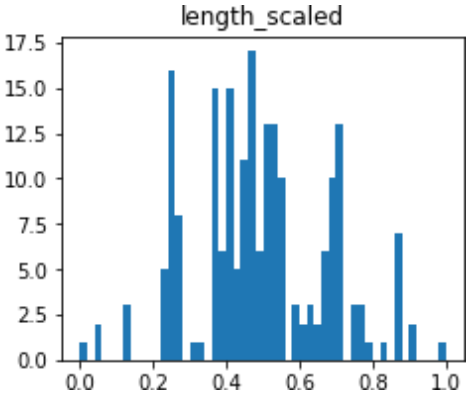

drive
120
100
80
60
40
20
0
0.0
0.5
1.0
1.5
2.0

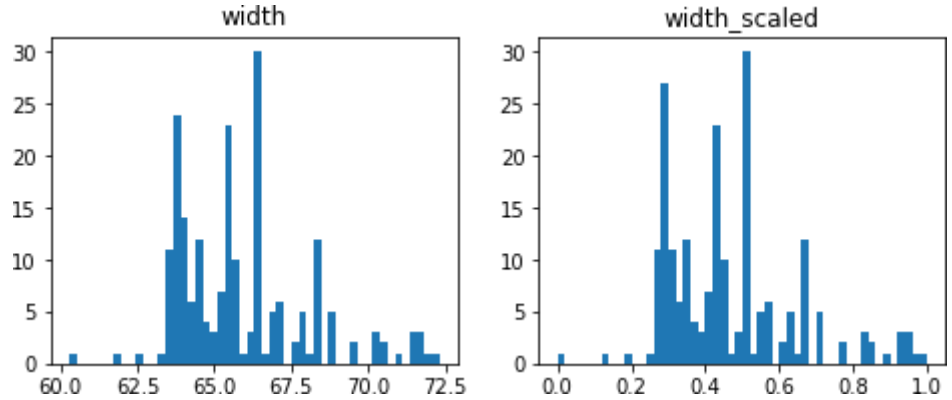

drive_scaled
120
100
80
60
40
20
0
0.0
0.2
0.4
0.6
0.8
1.0

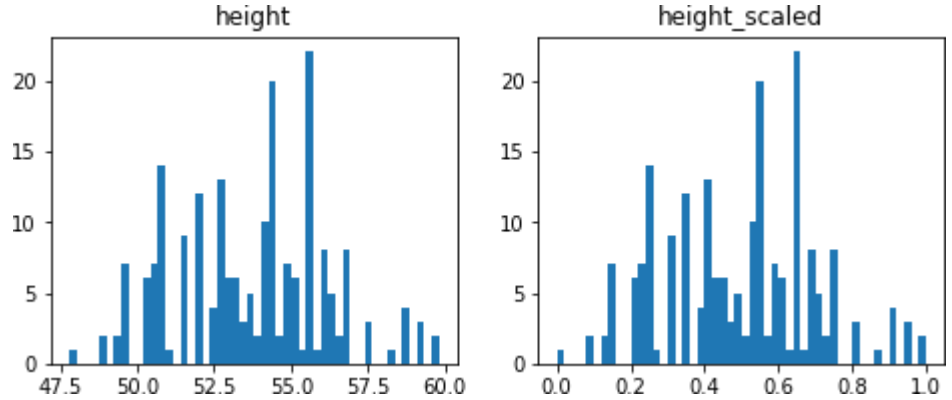

wheelbase
20
15
10
5
0
90
100
110
120

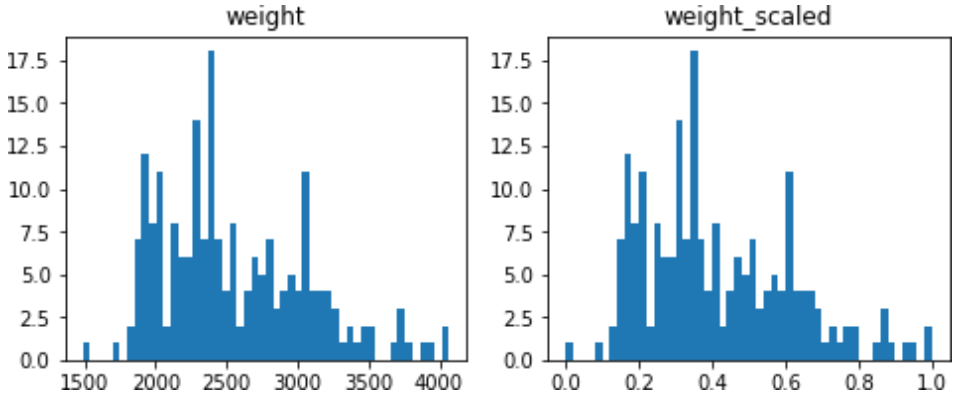

wheelbase_scaled
20
15
10
5
0
0.0
0.2
0.4
0.6
0.8
1.0

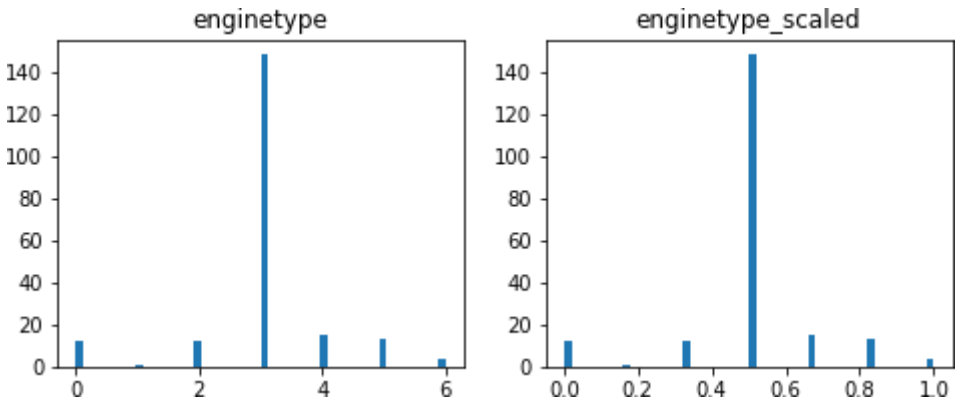

length
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
140
160
180
200


length_scaled
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0.0
0.2
0.4
0.6
0.8
1.0

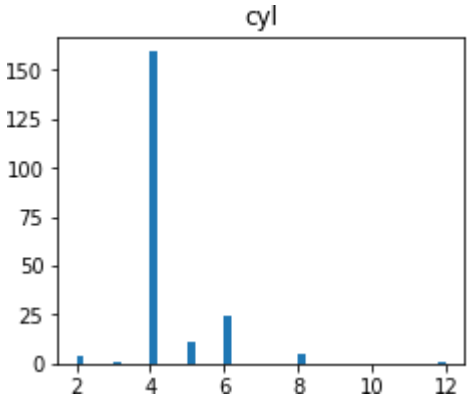

width
width_scaled

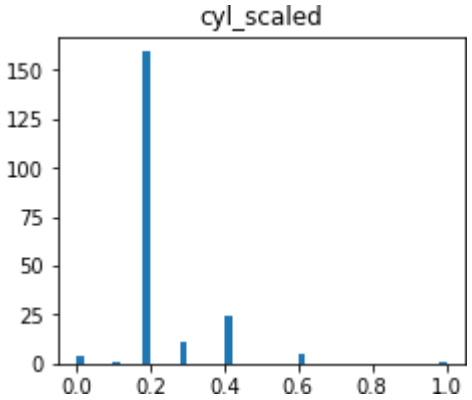

height
height_scaled

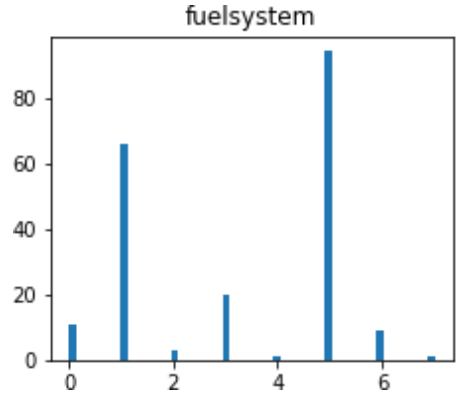

weight
weight_scaled

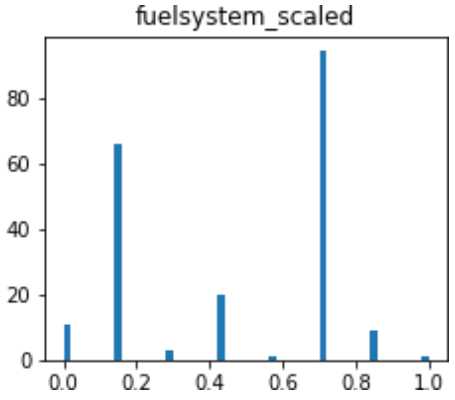

enginetype
140
120
100
80
60
40
20
0
0
2
4
6

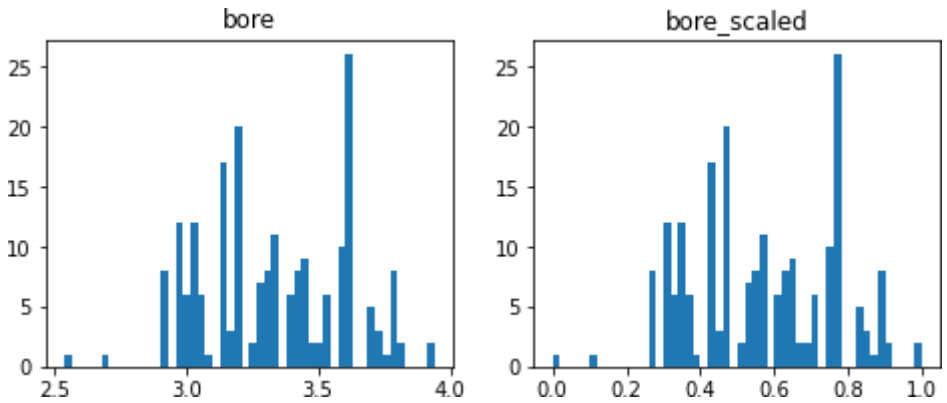
enginetype_scaled
140
120
100
80
60
40
20
0
0.0
0.2
0.4
0.6
0.8
1.0

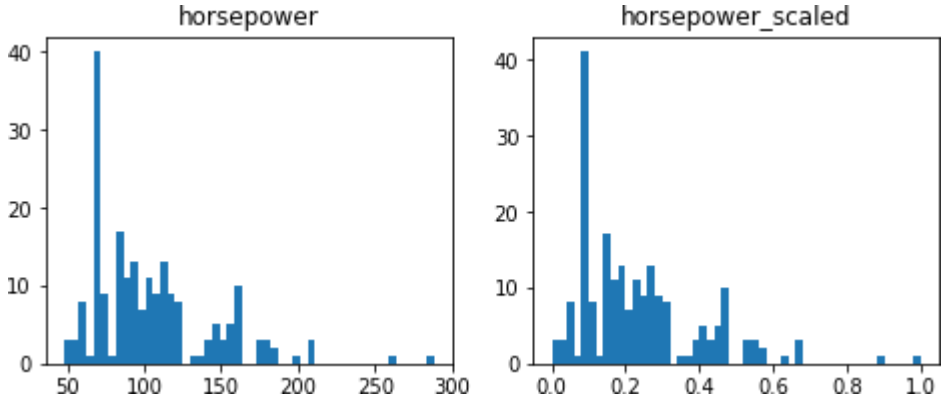

cyl
150
125
100
75
50
25
0
2
4
6
8
10
12

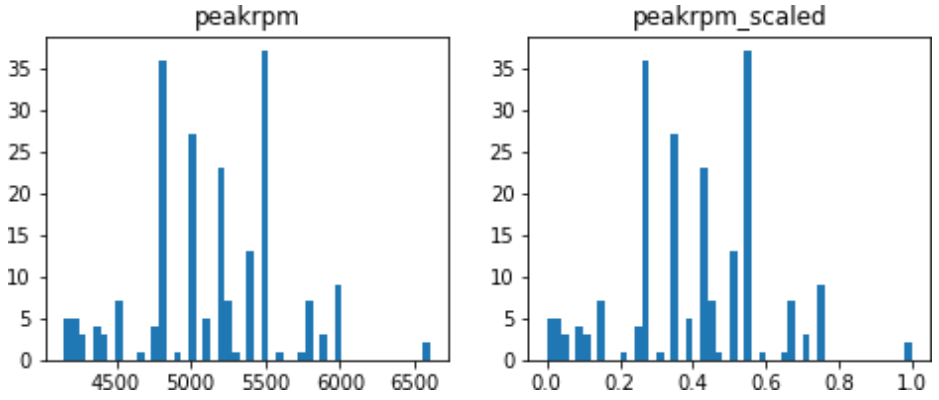

cyl_scaled
150
125
100
75
50
25
0
0.0
0.2
0.4
0.6
0.8
1.0

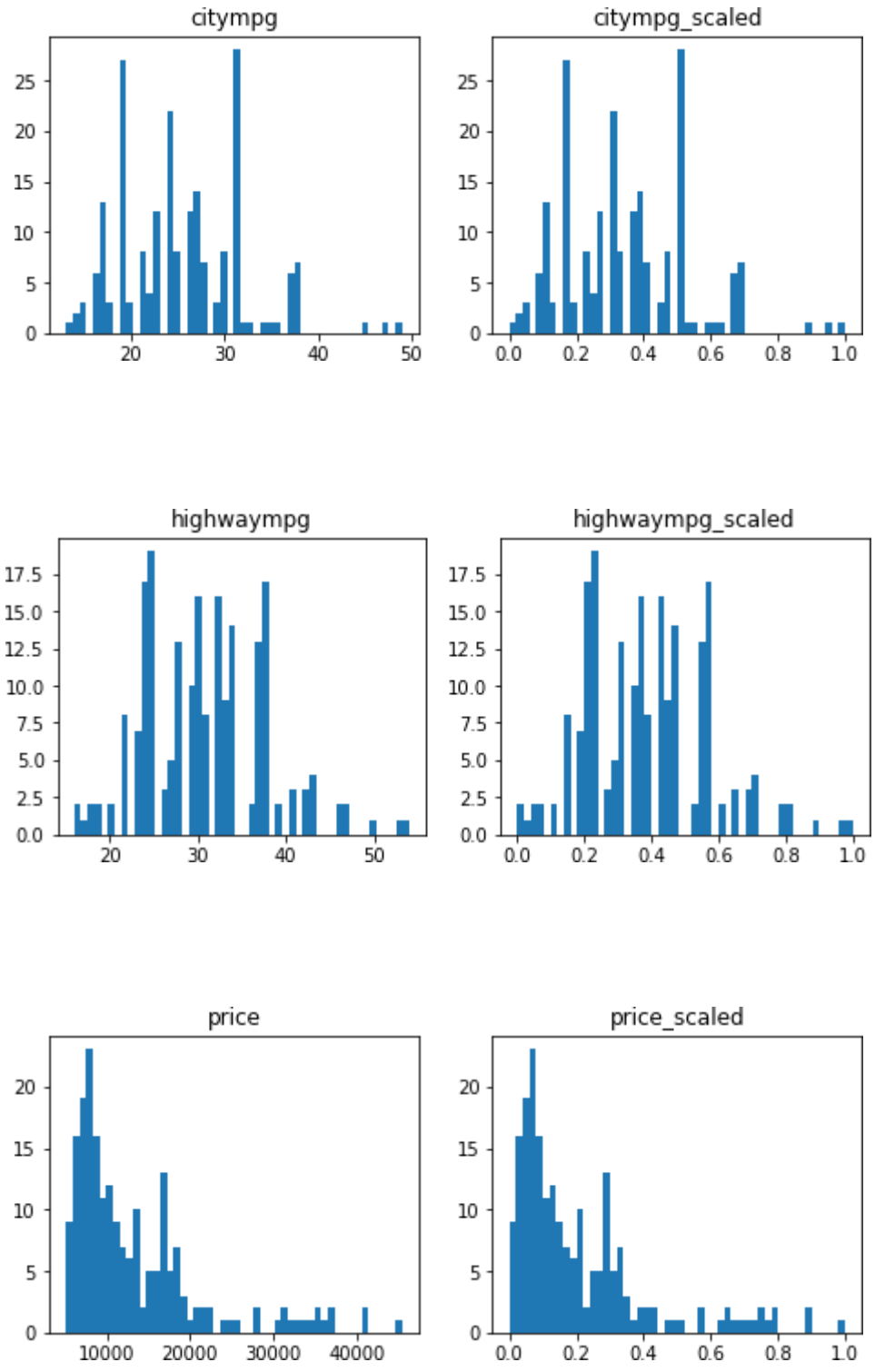

fuelsystem
80
60
40
20
0
0
2
4
6


fuelsystem_scaled
80
60
40
20
0
0.0
0.2
0.4
0.6
0.8
1.0


bore
bore_scaled


horsepower
horsepower_scaled

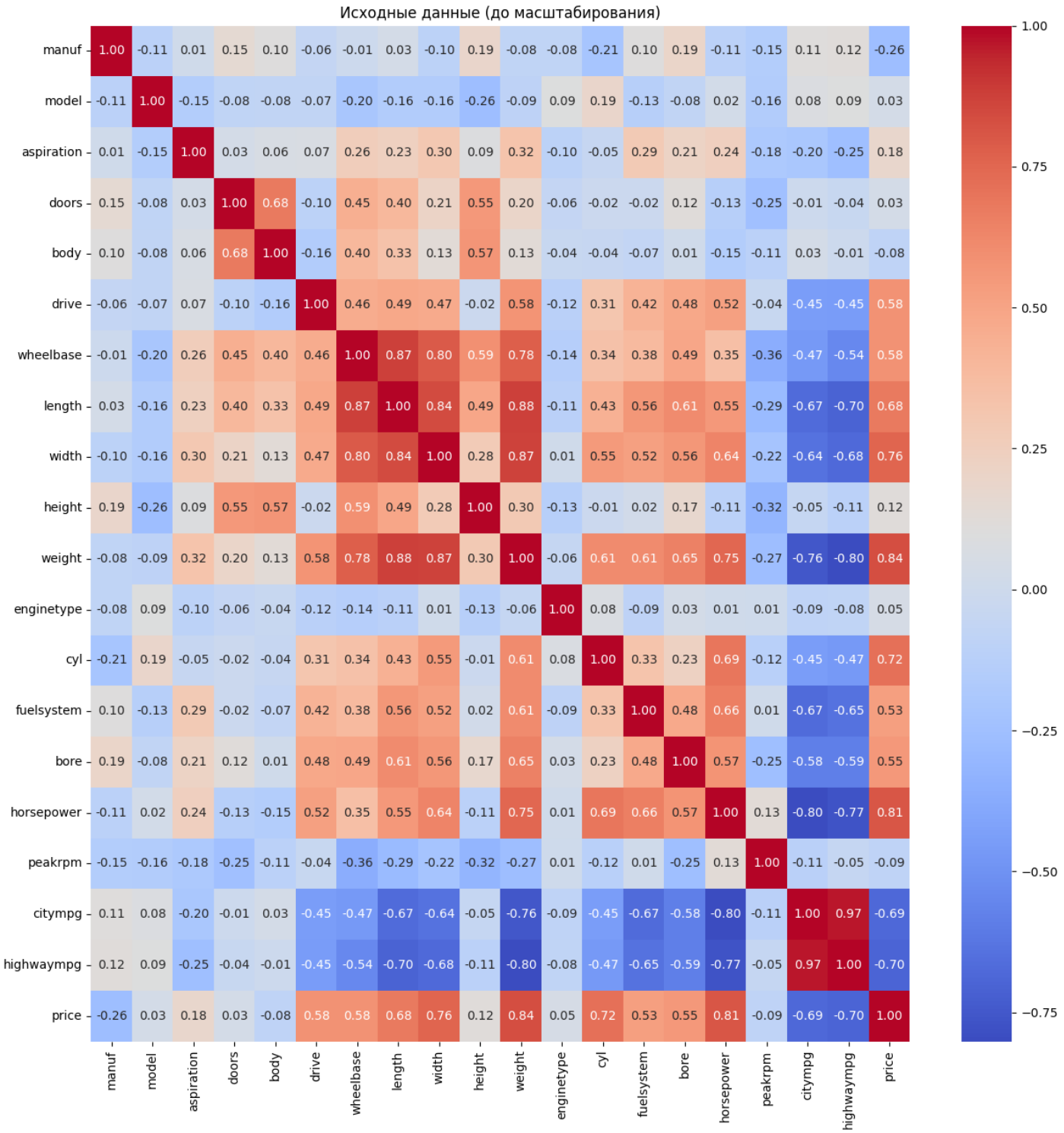

peakrpm
peakrpm_scaled



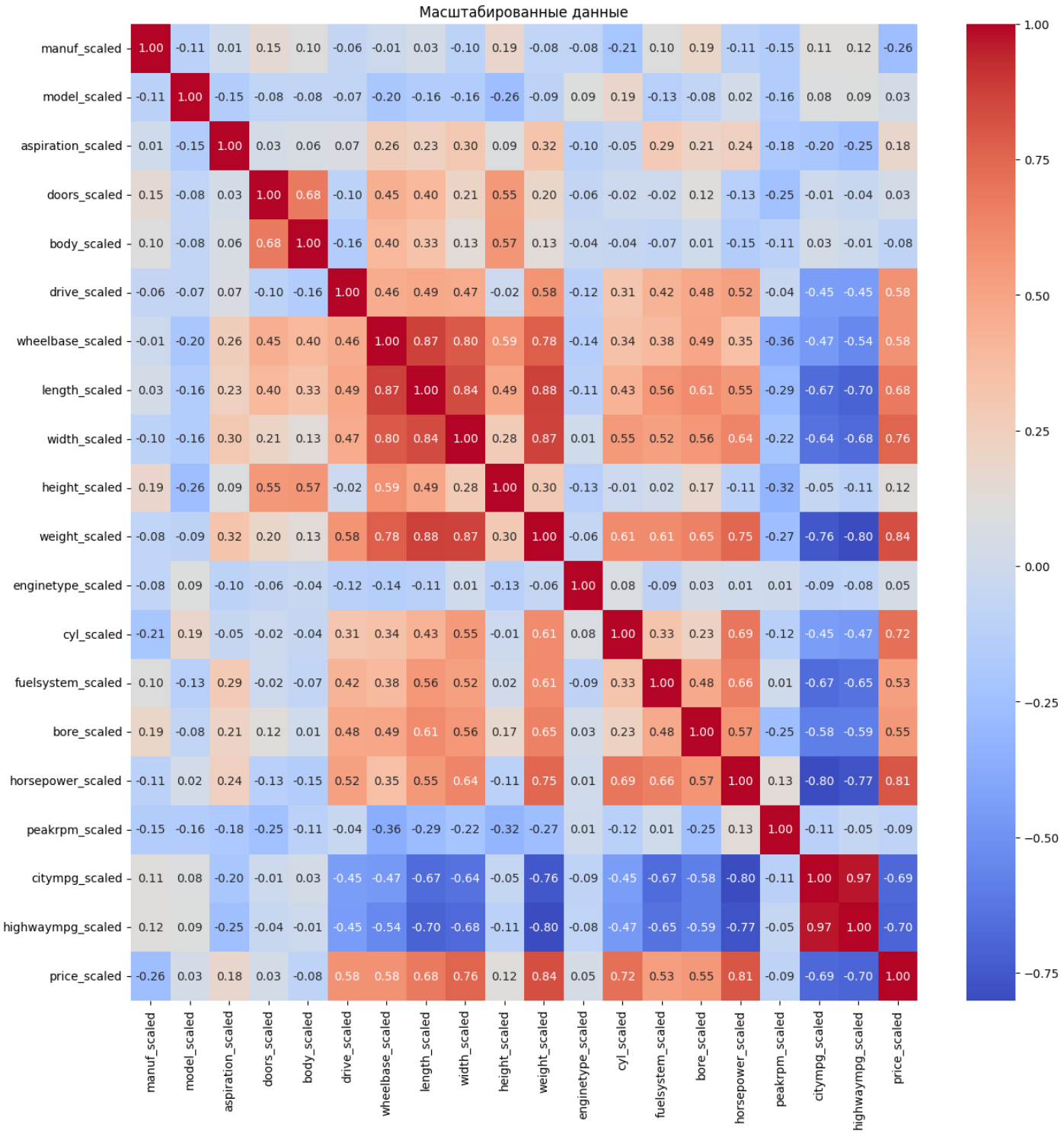
2.5. Корреляционный анализ данных

Построим корреляционные матрицы:

[45]:
```
fig, ax = plt.subplots(figsize=(15,15))
sns.heatmap(data[data.columns].corr(), annot=True, fmt='.2f', cmap=DYlGnBuD)
ax.set_title('Исходные данные (до масштабирования)')
plt.show()
```



[46]:
```
fig, ax = plt.subplots(figsize=(15,15))
sns.heatmap(data_scaled[data_scaled.columns].corr(), annot=True, fmt='.2f',
↪cmap=DY1GnBuD)
ax.set_title('Масштабированные данные')
plt.show()
```



На основании корреляционных матрицы можно сделать следующие выводы:

- Корреляционные матрицы для исходных и масштабированных данных идентичны
- Целевой признак регрессии “price” наиболее сильно коррелирует с “drive” (0.58), “wheelbase” (0.58), “length” (0.68), “width” (0.76), “weight” (0.84), “cyl” (0.72) и “horsepower” (0.81). Эти признаки в модели регрессии оставляем
- Признаки “citympg” и “highwaympg” имеют корреляцию, близкую по модулю к 1, поэтому оставим только один из них - “citympg”
- Данные позволяют построить модель машинного обучения

Удалим ненужный признак:

[47]:
```
data.drop(['highwaympg'], inplace=True, axis=1)
data_scaled.drop(['highwaympg_scaled'], inplace=True, axis=1)
```

[48]:
```
data.head()
```

[48]:

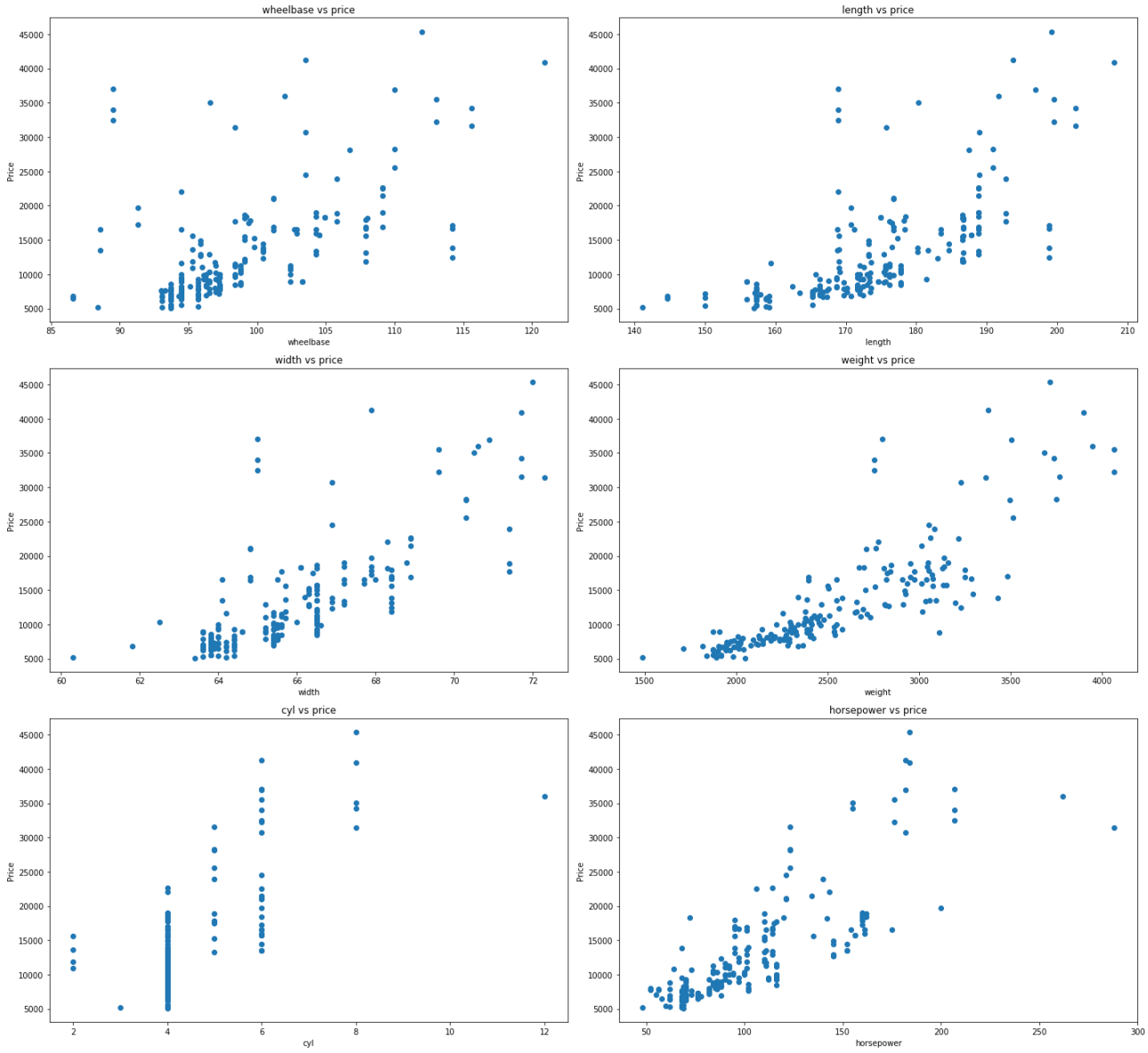
	manuf	model	aspiration	doors	body	drive	wheelbase	length	width	\
0	0	78	0	2	0	2	88.6	168.8	64.1	
1	0	122	0	2	0	2	88.6	168.8	64.1	
2	0	28	0	2	2	2	94.5	171.2	65.5	
3	1	0	0	4	3	1	99.8	176.6	66.2	
4	1	1	0	4	3	0	99.4	176.6	66.4	

	height	weight	enginetype	cyl	fuelsystem	bore	horsepower	peakrpm	\
0	48.8	2548	0	4	5	3.47	111	5000	
1	48.8	2548	0	4	5	3.47	111	5000	
2	52.4	2823	5	6	5	2.68	154	5000	
3	54.3	2337	3	4	5	3.19	102	5500	
4	54.3	2824	3	5	5	3.19	115	5500	

	citympg	price
0	21	13495.0
1	21	16500.0
2	19	16500.0
3	24	13950.0
4	18	17450.0

Построим графики зависимостей признаков с сильной корреляцией:

[49]:
```
def scatter(x,fig):
    plt.subplot(5,2,fig)
    plt.scatter(data[x],data['price'])
    plt.title(x+' vs price')
    plt.ylabel('Price')
    plt.xlabel(x)

plt.figure(figsize=(20,30))

scatter('wheelbase', 1)
scatter('length', 2)
scatter('width', 3)
scatter('weight', 4)
scatter('cyl', 5)
scatter('horsepower', 6)

plt.tight_layout()
```



2.6. Выбор подходящих моделей для решения задачи регрессии

Для решения задачи регрессии будем использовать следующие модели:

- Линейная регрессия
- Модель ближайших соседей
- Модель опорных векторов
- Дерево решений
- Случайный лес
- Градиентный бустинг

2.7. Выбор метрик для оценки качества моделей

В качестве метрик для решения задачи регрессии будем использовать метрики:

- Mean absolute error (средняя абсолютная ошибка)
- Mean squared error (средняя квадратичная ошибка)
- R2-score (коэффициент детерминации)

Они помогут определить качество моделей.
Метрики будем сохранять в класс:

[50]:
```
class MetricLogger:

    def __init__(self):
        self.df = pd.DataFrame(
            {'metric': pd.Series([], dtype='str'),
            'alg': pd.Series([], dtype='str'),
            'value': pd.Series([], dtype='float')})

    def add(self, metric, alg, value):
        """
        Добавление значения
        """
        # Удаление значения если оно уже было ранее добавлено
        self.df.drop(self.df[(self.df['metric']==metric)&(self.df['alg']==alg)].
↪index, inplace = True)
        # Добавление нового значения
        temp = [{'metric':metric, 'alg':alg, 'value':value}]
        self.df = self.df.append(temp, ignore_index=True)

    def get_data_for_metric(self, metric, ascending=True):
        """
        Формирование данных с фильтром по метрике
        """
        temp_data = self.df[self.df['metric']==metric]
        temp_data_2 = temp_data.sort_values(by='value', ascending=ascending)
        return temp_data_2['alg'].values, temp_data_2['value'].values

    def plot(self, str_header, metric, ascending=True, figsize=(5, 5)):
        """
        Вывод графика
        """
        array_labels, array_metric = self.get_data_for_metric(metric, ascending)
        fig, ax1 = plt.subplots(figsize=figsize)
        pos = np.arange(len(array_metric))
        rects = ax1.barh(pos, array_metric,
                         align='center',
                         height=0.5,
                         tick_label=array_labels)
        ax1.set_title(str_header)
        for a,b in zip(pos, array_metric):
            plt.text(0.5, a-0.05, str(round(b,3)), color='white')
        plt.show()
```

2.8. Формирование обучающей и тестовой выборок

Разделим выборку:

[51]:
```
from sklearn.model_selection import train_test_split
```

[52]:
```
X_train, X_test, y_train, y_test = train_test_split(data, data.price,
↪random_state=1)
```

[53]:
```
X_train.shape, y_train.shape, X_test.shape, y_test.shape
```

[53]:
```
((153, 19), (153,), (52, 19), (52,))
```

2.9. Построение базового решения (baseline) без подбора гиперпараметров

Построим базовые модели:

[54]:
```
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
```

[55]:
```
regr_models = {'LR': LinearRegression(),
               'KNN_20':KNeighborsRegressor(n_neighbors=20),
               'SVR':SVR(),
               'Tree':DecisionTreeRegressor(),
               'RF':RandomForestRegressor(),
               'GB':GradientBoostingRegressor()}
```

Сохраним метрики:

[56]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

[57]:
```
regrMetricLogger = MetricLogger()
```

[58]:
```
def regr_train_model(model_name, model, regrMetricLogger):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    mae = mean_absolute_error(y_test, y_pred)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    regrMetricLogger.add('MAE', model_name, mae)
    regrMetricLogger.add('MSE', model_name, mse)
    regrMetricLogger.add('R2', model_name, r2)

    print('{} \t MAE={}, MSE={}, R2={}'.format(
        model_name, round(mae, 3), round(mse, 3), round(r2, 3)))
```

Отобразим метрики:

[59]:
```
for model_name, model in regr_models.items():
    regr_train_model(model_name, model, regrMetricLogger)
```

```
LR       MAE=0.0, MSE=0.0, R2=1.0
KNN_20   MAE=769.6, MSE=5081904.089, R2=0.924
SVR      MAE=5494.731, MSE=70686472.641, R2=-0.053
Tree     MAE=364.109, MSE=875873.509, R2=0.987
RF       MAE=201.948, MSE=283052.672, R2=0.996
GB       MAE=140.891, MSE=126015.415, R2=0.998
```

Чем ближе значение MAE и MSE к 0 и R2 к 1 - тем лучше качество регрессии.

Видно, что по трем метрикам лучшая модель регрессии - у линейной модели. Но также по метрике R2-score модели градиентного бустинга, случайного леса и ближайших соседей близки к линейной.

Худшая модель по всем трем метрикам - модель опорных векторов.

2.10. Подбор оптимальной модели и гиперпараметра

Подберем оптимальные гиперпараметры:

[60]:
```
from sklearn.model_selection import GridSearchCV
```

[61]:
```
n_range = np.array(range(5,100,5))
tuned_parameters = [{'n_neighbors': n_range}]
tuned_parameters
```

[61]:
```
[{'n_neighbors': array([ 5,  10,  15,  20,  25,  30,  35,  40,  45,  50,  55,  60,  65,  70,
75,  80,  85,
         90,  95])}]
```

[62]:
```
%%time
regr_gs = GridSearchCV(KNeighborsRegressor(), tuned_parameters, cv=5,
 ↪scoring='neg_mean_absolute_error')
regr_gs.fit(X_train, y_train)
```

```
CPU times: total: 359 ms
Wall time: 365 ms
```

[62]:
```
GridSearchCV(cv=5, estimator=KNeighborsRegressor(),
             param_grid=[{'n_neighbors': array([ 5,  10,  15,  20,  25,  30,  35,  40,
45,  50,  55,  60,  65,  70,  75,  80,  85,
       90,  95])}],
             scoring='neg_mean_absolute_error')
```

Лучшая модель:

[63]:
```
regr_gs.best_estimator_
```

[63]:
```
KNeighborsRegressor()
```

Лучшее значение параметров:

[64]:
```
regr_gs.best_params_
```

[64]:
```
{'n_neighbors': 5}
```

Сохраним значение:

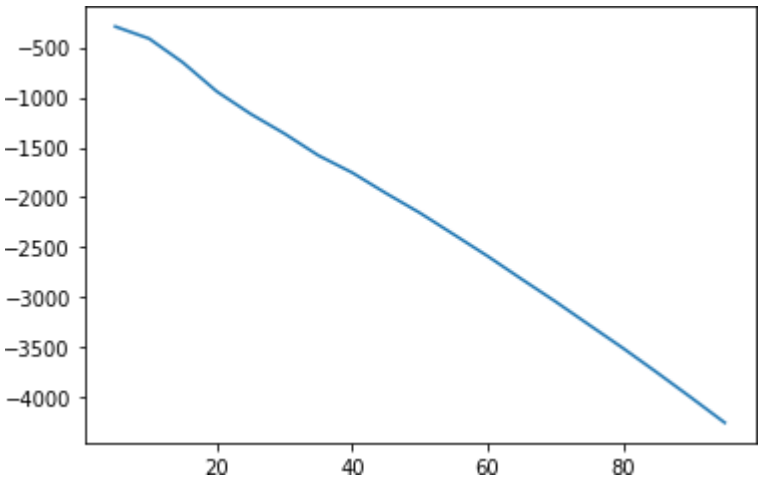
[65]:
```
regr_gs_best_params_txt = str(regr_gs.best_params_['n_neighbors'])
regr_gs_best_params_txt
```

[65]:
```
'5'
```

Изменение качества:

[66]:
```
plt.plot(n_range, regr_gs.cv_results_['mean_test_score'])
```

[66]:
```
[<matplotlib.lines.Line2D at 0x22a2b3696a0>]
```



2.11. Оптимальное значение гиперпараметра. Сравнение качества с baseline

Оптимальная модель - KNeighborsRegressor. Оптимальное значение гиперпараметра - 5. Сравним метрики с baseline моделью:

[67]:
```
regr_models_grid = {'KNN_20':KNeighborsRegressor(n_neighbors=20),
                    str('KNN_'+regr_gs_best_params_txt):regr_gs.best_estimator_}
```

[68]:
```
for model_name, model in regr_models_grid.items():
    regr_train_model(model_name, model, regrMetricLogger)
```

```
KNN_20    MAE=769.6, MSE=5081904.089, R2=0.924
KNN_5     MAE=263.517, MSE=417700.95, R2=0.994
```

Видим, что у оптимальной модели лучше качество, чем у исходной baseline-модели.

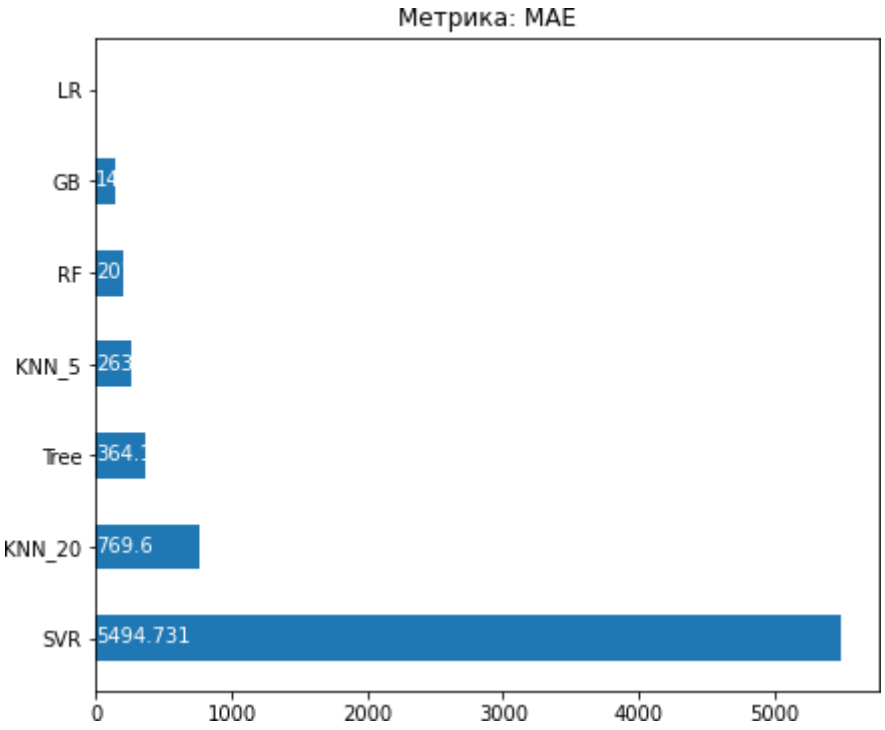
2.12. Формирование выводов о качестве построенных моделей

Сравним все метрики.

[69]:
```
regr_metrics = regrMetricLogger.df['metric'].unique()
```

Метрика Mean Absolute Error:

[70]:
```
regrMetricLogger.plot('Метрика: ' + 'MAE', 'MAE', ascending=False, figsize=(7, 6))
```



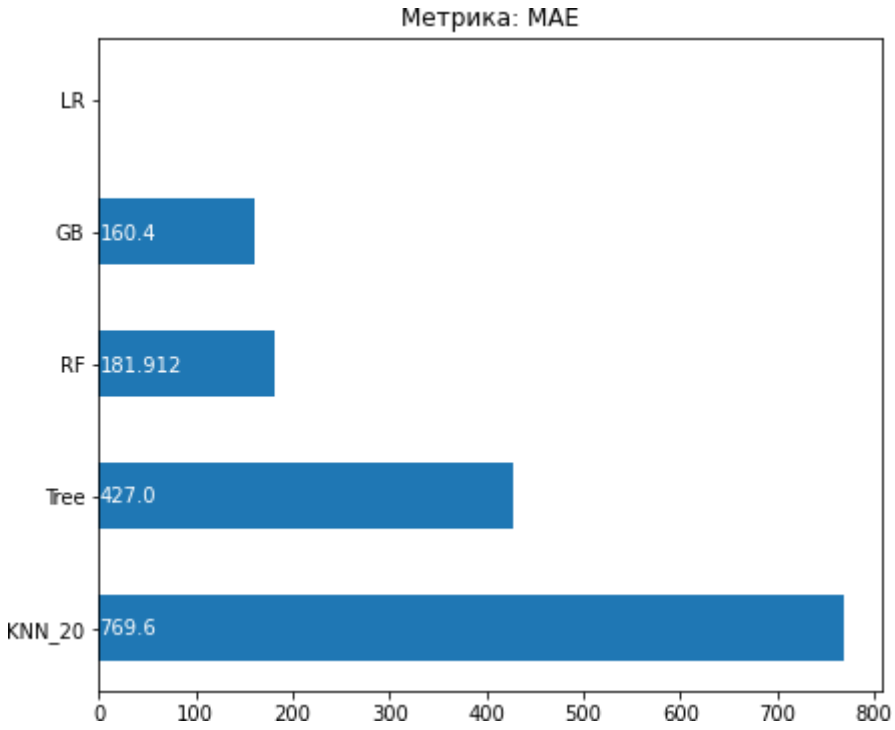
[71]:
```
regrMetricLogger_no_svr = MetricLogger()
```

[72]:
```
regr_models_no_svr = {'LR': LinearRegression(),
                      'KNN_20':KNeighborsRegressor(n_neighbors=20),
                      'Tree':DecisionTreeRegressor(),
                      'RF':RandomForestRegressor(),
                      'GB':GradientBoostingRegressor()}
```

[73]:
```
for model_name, model in regr_models_no_svr.items():
    regr_train_model(model_name, model, regrMetricLogger_no_svr)
```

```
LR        MAE=0.0, MSE=0.0, R2=1.0
KNN_20    MAE=769.6, MSE=5081904.089, R2=0.924
Tree      MAE=427.0, MSE=1365990.385, R2=0.98
RF        MAE=181.912, MSE=182933.855, R2=0.997
GB        MAE=160.4, MSE=166219.366, R2=0.998
```

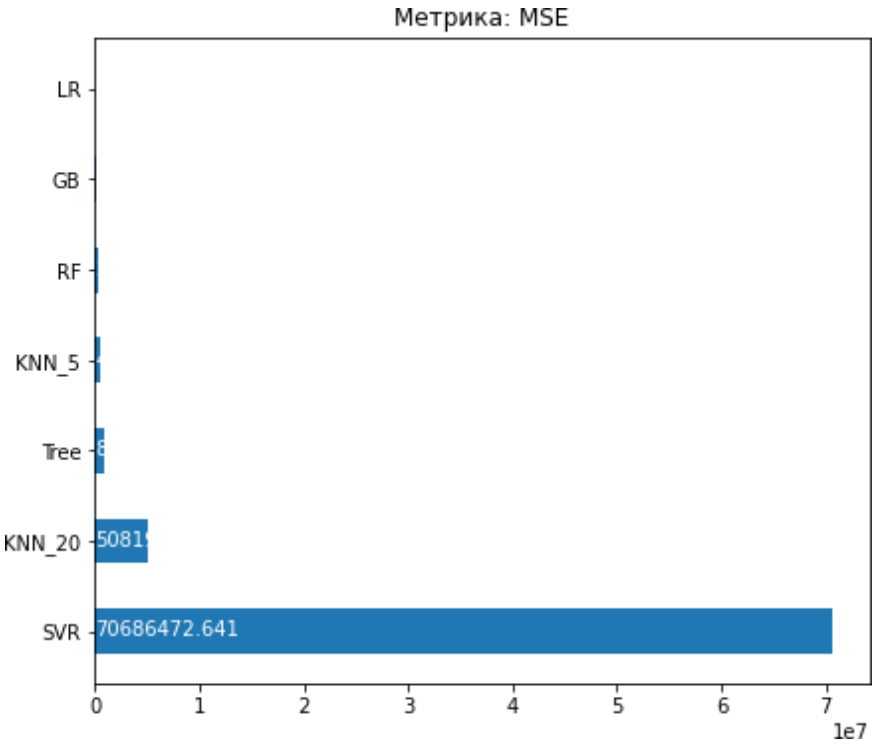
[74]:
```
regrMetricLogger_no_svr.plot('Метрика: ' + 'MAE', 'MAE', ascending=False,
 ↪figsize=(7, 6))
```



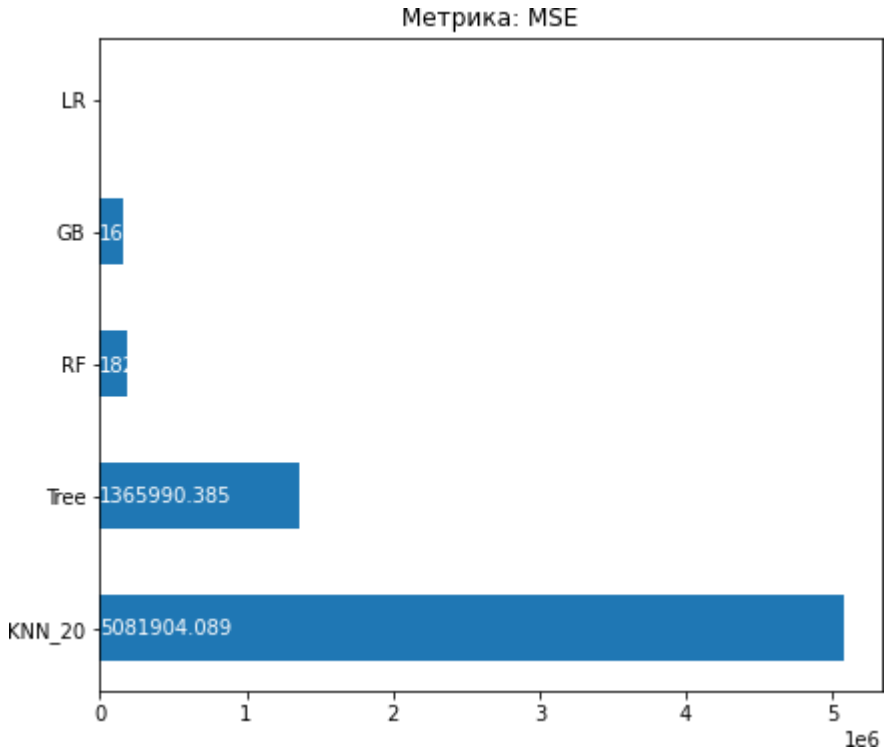
Чем ближе значение метрики к 0, тем качественне модель. Лучший результат показвывает модель линейной регрессии, худший - модель опорных векторов.

Метрика Mean Squarred Error:

[75]:
```
regrMetricLogger.plot('Метрика: ' + 'MSE', 'MSE', ascending=False, figsize=(7, 6))
```

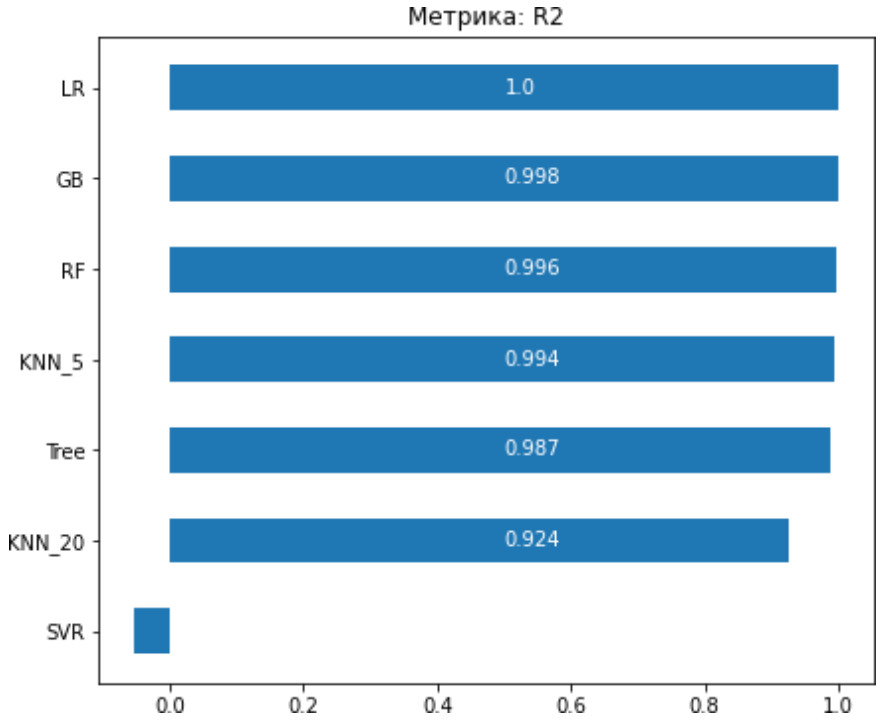


[76]:
```
regrMetricLogger_no_svr.plot('Метрика: ' + 'MSE', 'MSE', ascending=False,
↪figsize=(7, 6))
```

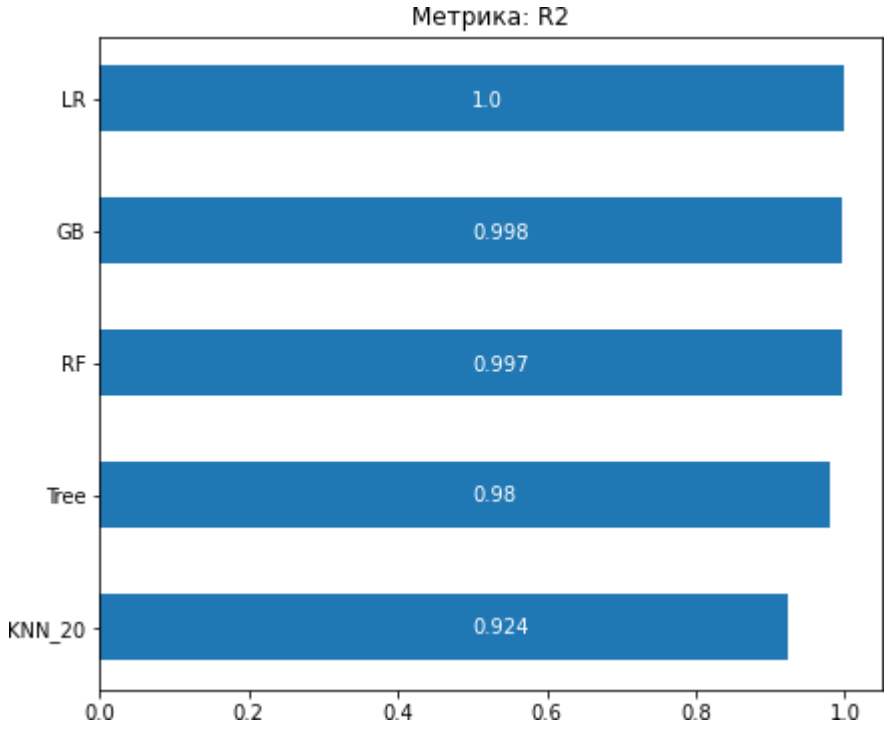


Чем ближе значение метрики к нулю, тем модель более качественна. Модель линейной регрессии выигрывает по качеству у остальных. Модель SVR обладает наихудшем качеством.

[77]:
```
regrMetricLogger.plot('Метрика: ' + 'R2', 'R2', ascending=True, figsize=(7, 6))
```



[78]:
```
regrMetricLogger_no_svr.plot('Метрика: ' + 'R2', 'R2', ascending=True,
↪figsize=(7, 6))
```



Исходя из метрики R2-score - наихудший результат показывает модель опорных векторов. Лучшими моделями можно считать модели линейной регрессии, градиентного бустинга, случайного леса и дерева решений.

Подводя итог: наиболее качественной моделью можно считать модель линейной регрессии.

3. Заключение

В работе был проведен разведочный анализ данных с обработкой данных с неинформативными признаками, пропусков и модификацией структуры и самих данных. Также было проведено кодирование категориальных признаков, масштабирование данных и сравнение масштабированных данных с исходными. Был выполнен корреляционный анализ и на его основании были выбраны модели для решения задачи регрессии. Исходные данные были разделены на тестовую и обучающую выборку, на основе этих выборок были обучены выбранные модели. Также была построена наиболее оптимальная модель. Все модели подверглись сравнению для определения наилучшего качества решения задачи регрессии, для этого использовались несколько метрик регрессии.

4. Список литературы

1. Kaggle: Your home for Data Science [Электронный ресурс]. URL: https://www.kaggle.com/
2. sckit-learn: machine learning in Python [Электронный ресурс]. URL: https://scikit-learn.org/stable/
3. Matplolib - visualization via Python [Электронный ресурс]. URL: https://matplotlib.org/
4. Методические указания по разработке НИРС [Электронный ресурс]. URL: https://github.com/ugapanyuk/courses_current/wiki/TMO_NIRS
5. Репозиторий курсов "Технологии машинного обучения", бакалавриат, 6 семестр [Электронный ресурс]. URL:https://github.com/ugapanyuk/courses_current/wiki/COURSE_TMO_SPRING_2023/