

Допустим, у нас есть друг по имени Том.

И Том хочет продать свою машину.

Но проблема в том, что он не знает, за сколько он должен продать свою машину.

Том хочет продать свою машину как можно дороже.

Но он также хочет установить разумную цену, чтобы кто-то захотел ее купить.

Поэтому цена, которую он назначит, должна отражать стоимость автомобиля.

Как мы можем помочь Тому определить наилучшую цену за его автомобиль?

Давайте подумаем, как ученые, изучающие данные, и четко определим некоторые из его проблем.

Например, есть ли данные о ценах на другие автомобили и их характеристиках?

Какие характеристики автомобилей влияют на их цены? Цвет? Марка? Влияет ли лошадиная сила на цену продажи, или, может быть, что-то другое?

Как аналитик данных или специалист по исследованию данных, это некоторые из вопросов, над которыми мы можем начать думать. Чтобы ответить на эти вопросы, нам понадобятся некоторые данные.

Смотрим данные

Each of the attributes in the dataset

No.	Attribute name	attribute range	No.	Attribute name	attribute range
1	symboling	-3, -2, -1, 0, 1, 2, 3.	14	curb-weight	continuous from 1488 to 4066.
2	normalized-losses	continuous from 65 to 256.	15	engine-type	dohc, dohcvt, l, ohc, ohcf, ohcv, rotor.
3	make	audi, bmw, etc.	16	num-of-cylinders	eight, five, four, six, three, twelve, two.
4	fuel-type	diesel, gas.	17	engine-size	continuous from 61 to 326.
5	aspiration	std, turbo.	18	fuel-system	1bbl, 2bbl, 4bbl, idi, mfi, mpfi, spdi, spfi.
6	num-of-doors	four, two.	19	bore	continuous from 2.54 to 3.94.
7	body-style	hardtop, wagon, etc.	20	stroke	continuous from 2.07 to 4.17.
8	drive-wheels	4wd, fwd, rwd.	21	compression-ratio	continuous from 7 to 23.
9	engine-location	front, rear.	22	horsepower	continuous from 48 to 288.
10	wheel-base	continuous from 86.6 to 120.9.	23	peak-rpm	continuous from 4150 to 6600.
11	length	continuous from 141.1 to 208.1.	24	city-mpg	continuous from 13 to 49.
12	width	continuous from 60.3 to 72.3.	25	highway-mpg	continuous from 16 to 54.
13	height	continuous from 47.8 to 59.8.	26	price	continuous from 5118 to 45400.

Первый атрибут, обозначение, соответствует уровню страхового риска автомобиля.

Первоначально автомобилям присваивается символ фактора риска, связанный с их ценой.

Затем, если автомобиль является более рискованным, этот символ корректируется, перемещаясь вверх по шкале.

Значение плюс три означает, что автомобиль является рискованным.

Минус три - это, вероятно, довольно безопасный автомобиль.

Второй атрибут, нормализованные убытки, это относительная средняя выплата по убыткам за год страхования автомобиля.

Это значение нормируется для всех автомобилей в пределах определенной классификации размеров, двухдверные малые, универсалы, спортивные специализированные, и т.д., и представляет собой средний убыток на автомобиль в год.

Значения вариируются от 65 до 256. Итак, после того как мы поймем значение каждого признака,

мы заметим, что 26 атрибутом является цена.

Это наше целевое значение или, другими словами, метка.

Это означает, что цена - это значение, которое мы хотим предсказать из набора данных, а

предикторами должны быть все остальные переменные, перечисленные в списке, такие как,

нормализованные потери, марка и так далее.

Таким образом, целью данного проекта является предсказание цены по другим характеристикам автомобиля.

БИБЛИОТЕКИ ПИТОН

Основным инструментом Pandas является двумерная таблица, состоящая из столбцов

и метки строк, которая называется фреймом данных. Он предназначен для обеспечения простоты индексирования.

Библиотека NumPy использует массивы для своих входов и выходов. Она может быть расширена до объектов для матриц и С небольшими изменениями в кодировке разработчики могут выполнять быструю обработку массивов.

SciPy включает функции для некоторых сложных математических задач, перечисленных на этом слайде, а также для визуализации данных. Использование методов визуализации данных - лучший способ общения с другими людьми, показывая им значимые результаты анализа. Эти библиотеки позволяют создавать графики, диаграммы и карты.

Пакет **Matplotlib** - самая известная библиотека для визуализации данных. Он отлично подходит для построения графиков и диаграмм. Графики также хорошо настраиваются.

Еще одна библиотека визуализации высокого уровня - **Seaborn**. Она основана на **Matplotlib**. С ее помощью очень легко создавать различные графики, такие как тепловые карты, временные ряды и скрипичные графики.

С помощью алгоритмов машинного обучения, мы можем разработать модель, используя наш набор данных, и получить прогнозы.

Здесь мы представляем два пакета, библиотека **Scikit-learn** содержит инструменты статистического моделирования, включая регрессию, классификацию, кластеризацию и так далее. Эта библиотека построена на NumPy, SciPy и Matplotlib.

Statsmodels - это также модуль Python, который позволяет пользователям исследовать данные, оценивать статистические модели и выполнять статистические тесты.

Importing and Exporting Data in Python

Чтение данных в pandas можно быстро выполнить в три строки. Сначала импортируйте pandas, затем определите переменную с путем к файлу, а затем используйте метод read_CSV для импорта данных.

Import pandas as pd # где pd это псевдоним

```
url = "ссылка на источник" # где url - это переменная, которая будет хранить ссылку на источник данных
```

```
df = pd.read_csv(url) # data frame – переменная, в которую положат прочитанный файл
```

Если в файле нет заголовка таблицы то

```
df = pd.read_csv(url, header = None)
```

Поскольку печать (**df**) всего набора данных может занять слишком много времени и ресурсов, чтобы сэкономить время, мы можем просто использовать **dataframe.head**, чтобы показать первые n строк фрейма данных. Аналогично, **dataframe.tail** показывает последние строки фрейма данных.

Printing the dataframe in Python

- `df` prints the entire dataframe (not recommended for large datasets)
- `df.head(n)` to show the first n rows of data frame.
- `df.tail(n)` shows the bottom n rows of data frame.

df.head()

	0	1	2	3	4	5	6	7	8	9	...	16	17	18	19	20	21	22	23	24	25
0	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	111	5000	21	27	13495
1	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	111	5000	21	27	16500
2	1	?	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	9.0	154	5000	19	26	16500
3	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	10.0	102	5500	24	30	13950
4	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.40	8.0	115	5500	18	22	17450

Добавляем заголовки

Adding headers

- Replace default header (by `df.columns = headers`)

```
headers = ["symboling", "normalized-losses", "make", "fuel-type", "aspiration", "num-of-doors", "body-style",  
"drive-wheels", "engine-location", "wheel-base", "length", "width", "height", "curb-weight", "engine-type",  
"num-of-cylinders", "engine-size", "fuel-system", "bore", "stroke", "compression-ratio", "horsepower", "peak-  
rpm", "city-mpg", "highway-mpg", "price"]
```

```
df.columns=headers
```

```
df.head(5)
```

Adding headers

- Replace default header (by `df.columns = headers`)

symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke	compression-ratio	h	
0	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	1
1	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	1
2	1	?	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	9.0	1
3	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	10.0	1
4	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.40	8.0	1

Дальше мы можем экспортовать данные в новый файл

Exporting a Pandas dataframe to CSV

- Preserve progress anytime by saving modified dataset using

```
path="C:/Windows/.../ automobile.csv"
```

```
df.to_csv(path)
```

Exporting to different formats in Python



Data Format	Read	Save
csv	pd.read_csv()	df.to_csv()
json	pd.read_json()	df.to_json()
Excel	pd.read_excel()	df.to_excel()
sql	pd.read_sql()	df.to_sql()

ИССЛЕДОВАНИЕ НАБОРА ДАННЫХ

В Pandas есть несколько встроенных методов, которые можно использовать для понимания:

- тип данных или характеристики
- распределение данных в наборе данных.
- Использование этих методов дает общее представление о наборе данных, а также указывает на потенциальные проблемы, такие как неправильный тип данных характеристики, которые, возможно, придется решать позже.

Данные имеют различные типы.

Основными типами, хранящимися в объектах Pandas, являются object, float, Int и datetime. Имена типов данных несколько отличаются от имен в родном Python.

В этой таблице показаны различия и сходства между ними. Некоторые из них очень похожи, например, числовые типы данных int и float. Объектная функция типа pandas похожа на строку в Python, за исключением изменения имени.

В то время как тип datetime Pandas, является очень полезным типом для работы с данными временных рядов.

Есть две причины проверять типы данных в наборе данных. Pandas автоматически назначает типы на основе на основе кодировки, которую он обнаруживает в исходной таблице данных.

По ряду причин, это назначение может быть неверным. Например, должно быть неловко, если столбец цены автомобиля, который, как мы должны ожидать, будет содержать непрерывные числовые числа, присвоен тип данных `object`.

Было бы более естественно, если бы он имел тип `float`. Джерри может понадобиться вручную изменить тип данных на `float`.

Вторая причина заключается в том, что позволяет опытным специалистам по работе с данными увидеть, какие функции Python могут быть применены к определенному столбцу.

Например, некоторые математические функции можно применять только к числовым данным. Если применить эти функции к нечисловым данным, может возникнуть ошибка.

Когда метод `dtype` применяется к набору данных, тип данных каждого столбца возвращается в виде серии.

Хорошая интуиция исследователя данных подсказывает нам, что большинство типов данных имеют смысл.

Basic Insights of Dataset - Data Types

- In pandas, we use `dataframe.dtypes` to check data types

```
df.dtypes
```

symboling	int64
normalized-losses	object
make	object
fuel-type	object
aspiration	object
num-of-doors	object
body-style	object
drive-wheels	object
engine-location	object
wheel-base	float64
length	float64
width	float64
height	float64
curb-weight	int64
engine-type	object
num-of-cylinders	object
engine-size	int64
fuel-system	object
bore	object
stroke	object
compression-ratio	float64
horsepower	object
peak-rpm	object
city-mpg	int64
highway-mpg	int64
price	object
	dtype: object

Например, марки автомобилей - это имена. Значит, эта информация должна быть типа `object`.

С последним в списке может возникнуть проблема.

Поскольку отверстие - это размер двигателя, мы должны ожидать, что будет использоваться числовой тип данных.

Вместо этого используется тип `object`. В последующих разделах Джерри придется исправлять эти несоответствия типов.

Теперь мы хотим проверить статистическую сводку по каждому столбцу, чтобы узнать о распределении данных в каждом столбце.

Статистические метрики могут подсказать специалисту по исследованию данных, если существуют математические проблемы, такие как экстремальные выбросы и большие отклонения.

Возможно, ученым придется решать эти проблемы позже.

Чтобы получить быструю статистику, мы используем метод `describe`.

dataframe.describe()

- Returns a statistical summary

```
df.describe()
```

	symboling	wheel-base	length	width	height	curb-weight	engine-size	compression-ratio	city-mpg	highway-mpg
count	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000
mean	0.834146	98.756585	174.048268	65.907905	53.724878	2555.565854	126.907917	10.142537	25.219512	30.751220
std	1.245307	6.021776	12.337289	2.145204	2.443522	520.680204	41.842693	3.972040	6.542142	6.886443
min	-2.000000	86.800000	141.100000	63.300000	47.800000	1488.000000	61.000000	7.000000	13.000000	16.000000
25%	0.000200	94.500000	168.300000	64.100000	52.000000	2145.000000	97.000000	8.600000	19.000000	25.000000
50%	1.000200	97.000000	173.200000	65.500000	54.100000	2414.000000	120.000000	9.000000	24.000000	30.000000
75%	2.000200	102.400000	183.100000	66.900000	56.500000	2935.000000	141.000000	9.400000	30.000000	34.000000
max	3.000000	120.900000	208.100000	72.300000	59.800000	4066.000000	326.000000	23.000000	49.000000	54.000000

Возвращает количество терминов в столбце как `count`, среднее значение столбца как `mean`, стандартное отклонение столбца как `std`, максимальные минимальные значения, а также границу каждого квартиля.

По умолчанию функция **dataframe.describe** пропускает строки и столбцы, не содержащие чисел.

Можно сделать так, чтобы метод **describe** работал и для столбцов объектного типа.

Чтобы включить суммирование всех столбцов, мы можем добавить аргумент.

Include equals all внутри скобки функции `describe`.

dataframe.describe(include="all")

- Provides full summary statistics
`df.describe(include="all")`

	symboling	normalized-losses	make	fuel-type	aspiration	normalized-size	body-style	drive-wheels	engine-location	wheel-base	-	engine-size	fuel-system	bore	stroke
count	205.000000	205	205	205	205	205	205	205	205	205.000000	-	205.000000	205	205	205
unique	NaN	52	22	2	2	3	5	3	2	NaN	-	NaN	8	39	37
top	NaN	7	Izuzu	gas	sld	low	ardan	front	front	NaN	-	NaN	top1	3.52	3.40
freq	NaN	41	22	106	106	114	95	123	202	NaN	-	NaN	94	103	20
mean	0.83149	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	98.246585	-	109.862317	NaN	NaN	NaN
std	1.352007	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	6.031779	-	14.643993	NaN	NaN	NaN
min	-2.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	65.000000	-	47.000000	NaN	NaN	NaN
25%	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	94.500000	-	97.000000	NaN	NaN	NaN
50%	1.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	97.000000	-	108.000000	NaN	NaN	NaN
75%	2.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	102.400000	-	141.000000	NaN	NaN	NaN
max	5.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	120.900000	-	326.000000	NaN	NaN	NaN

Теперь результат показывает сводку по всем 26 столбцам, включая атрибуты объектного типа. Мы видим, что для столбцов объектного типа оценивается другой набор статистических данных, таких как уникальность, топ и частота.

Уникальность - это количество различных объектов в столбце. Топ - это наиболее часто встречающийся объект, а freq - это количество раз, когда верхний объект появляется в столбце.

Некоторые значения в таблице показаны здесь как NaN, что означает "не число". Это связано с тем, что данная конкретная статистическая метрика не может быть рассчитана для этого конкретного типа данных столбца.

Другой метод, который вы можете использовать для проверки вашего набора данных, это функция **dataframe.info**. Эта функция показывает верхние 30 строк и нижние 30 строк фрейма данных.

Вот как типичный пользователь получает доступ к базам данных с помощью кода Python, написанного в блокноте Jupyter, веб-редакторе.

Существует механизм, с помощью которого программа Python взаимодействует с СУБД.

Python-код подключается к базе данных с помощью вызовов API. Мы объясним основы SQL API и Python DB API.

Интерфейс прикладного программирования - это набор функций, которые вы можете вызвать для получения доступа к какому-либо виду услуг. SQL API состоит из вызовов библиотечных функций в качестве интерфейса прикладного программирования, API, для СУБД.

Для передачи SQL-запросов в СУБД прикладная программа вызывает функции в API, а для получения запросов - другие функции. Вызывает другие функции для

получения результатов запроса и информации о состоянии СУБД. Принцип работы типичного SQL API показан на рисунке.

Прикладная программа начинает свой доступ к базе данных с одного или нескольких вызовов API, которые соединяют программу с СУБД.

Чтобы отправить SQL-запрос в СУБД, программа формирует запрос в виде текстовой строки в буфере, а затем делает вызов API для передачи содержимого буфера в СУБД.

Прикладная программа выполняет вызовы API для проверки состояния своего запроса к СУБД и для обработки ошибок.

Прикладная программа завершает свой доступ к базе данных вызовом API, который отключает ее от базы данных.

DB-API - это стандартный API Python для доступа к реляционным базам данных.

Этот стандарт позволяет вам написать одну программу, которая работает с несколькими реляционными базами данных, вместо того чтобы писать отдельную программу для каждой из них.

Таким образом, если вы изучите функции DB-API, то сможете применить эти знания для работы с любой базой данных с помощью Python.

Две основные концепции в Python DB-API - это объекты соединений и объекты запросов. Объекты соединений используются для подключения к базе данных и управления транзакциями.

Объекты курсора используются для выполнения запросов. Вы открываете объект курсора, а затем выполняете запросы. Курсор работает аналогично курсору в системе обработки текста, где вы прокручиваете вниз

в наборе результатов и получаете данные в приложении. Курсоры используются для сканирования результатов в базе данных. Вот методы, используемые с объектами соединения.

- **Метод cursor() возвращает новый объект курсора, использующий соединение.**
- **Метод commit() используется для фиксации любой ожидающей транзакции в базе данных.**
- **Метод rollback() заставляет базу данных откатиться к началу любой транзакции.**
- **Метод close() используется для закрытия соединения с базой данных.**

Давайте рассмотрим приложение Python, которое использует DB-API для запроса к базе данных.

Сначала вы импортируете свой модуль базы данных, используя API connect из этого модуля.

Writing code using DB-API

```
from dbmodule import connect

#Create connection object
connection = connect('databasename','username','pswd')

#Create a cursor object
cursor = connection.cursor()

#Run queries
cursor.execute('select * from mytable')
results = cursor.fetchall()

#Free resources
Cursor.close()
connection.close()
```

Чтобы открыть соединение с базой данных, вы используете функцию `connection` и передаете в нее

параметры, то есть имя базы данных, имя пользователя и пароль. **Функция `connect` возвращает объект соединения.**

После этого вы создаете объект курсора на объекте соединения. Курсор используется для выполнения запросов и получения результатов. После выполнения запросов с помощью курсора мы также используем курсор для получения результатов запроса.

Наконец, когда система завершает выполнение запросов, она освобождает все ресурсы, закрывая соединение. Помните, что всегда важно закрывать соединения, чтобы неиспользуемые соединения не занимали ресурсы.

ИТОГИ ПЕРВОГО МОДУЛЯ

Определить бизнес-проблему: посмотреть на данные и принять решение на высоком уровне о том, какой анализ необходимо провести.

Импортировать и экспортить данные в Python: Как импортировать данные из различных источников данных с помощью библиотеки Pandas и как экспортить файлы в различные форматы.

Анализ данных в Python: Как проводить вводный анализ в Python, используя такие функции, как `dataframe.head()` для просмотра первых нескольких строк набора данных, `dataframe.info()` для просмотра названий столбцов и типов данных.

УСТАНОВКА БИБЛИОТЕК

```
import piplite
```

```
import micropip  
  
await piplite.install(['pandas'])  
  
await piplite.install(['matplotlib'])  
  
await piplite.install(['scipy'])  
  
await piplite.install(['seaborn'])  
  
await micropip.install(['ipywidgets'],keep_going=True)  
  
await micropip.install(['tqdm'],keep_going=True)
```

ИМПОРТ ПАНДАС

```
import pandas as pd  
  
import numpy as np
```

Read Data

```
path = "https://cf-courses-data.s3.us.cloud-object-  
storage.appdomain.cloud/IBMDriverSkillsNetwork-DA0101EN-  
SkillsNetwork/labs/Data%20files/auto.csv"  
  
await download(path, "auto.csv")  
  
path="auto.csv"  
  
  
  
# Import pandas library  
  
import pandas as pd  
  
  
  
# Read the online file by the URL provides above, and assign it to variable "df"  
df = pd.read_csv(path, header=None)  
  
  
  
# show the first 5 rows using dataframe.head() method  
  
print("The first 5 rows of the dataframe")  
  
df.head(5) # выведет 5 строк с начала
```

```
The first 5 rows of the dataframe
   0   1   2   3   4   5   6   7   8   9   ...   16   17   18   19   20   21   22   23   24   25
0  3   ? alfa-romero  gas std two convertible rwd front 88.6 ...
1  3   ? alfa-romero  gas std two convertible rwd front 88.6 ...
2  1   ? alfa-romero  gas std two hatchback rwd front 94.5 ...
3  2  164      audi  gas std four sedan fwd front 99.8 ...
4  2  164      audi  gas std four sedan 4wd front 99.4 ...

5 rows × 26 columns
```

`print ('bottom 10 rows')`

`df.tail(10) # выведет 10 строк с конца`

```
bottom 10 rows
   0   1   2   3   4   5   6   7   8   9   ...   16   17   18   19   20   21   22   23   24   25
195 -1  74 volvo  gas std four wagon rwd front 104.3 ...
196 -2 103 volvo  gas std four sedan rwd front 104.3 ...
197 -1  74 volvo  gas std four wagon rwd front 104.3 ...
198 -2 103 volvo  gas turbo four sedan rwd front 104.3 ...
199 -1  74 volvo  gas turbo four wagon rwd front 104.3 ...
200 -1  95 volvo  gas std four sedan rwd front 109.1 ...
201 -1  95 volvo  gas turbo four sedan rwd front 109.1 ...
202 -1  95 volvo  gas std four sedan rwd front 109.1 ...
203 -1  95 volvo diesel turbo four sedan rwd front 109.1 ...
204 -1  95 volvo  gas turbo four sedan rwd front 109.1 ...

10 rows × 26 columns
```

Добавим заголовки столбцов

`# create headers list`

```
headers = ["symboling","normalized-losses","make","fuel-type","aspiration", "num-of-doors",
"body-style", "drive-wheels","engine-location","wheel-base",
"length","width","height","curb-weight","engine-type", "num-of-cylinders", "engine-size",
"fuel-system","bore","stroke","compression-ratio","horsepower", "peak-rpm","city-mpg",
"highway-mpg","price"]
```

`print("headers\n", headers)`

ВОПРОС 2: НАЙДИТЕ ИМЯ СТОЛБЦА ФРЕЙМА ДАННЫХ.

`df.columns = headers`

`df.head(10)`

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke	compression-ratio	horsepower	peak-rpm	city-mpg	highway-mpg	price
0	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	—	130	mpfi	3.47	2.68	9.0	111	5000	21	27	13495
1	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	—	130	mpfi	3.47	2.68	9.0	111	5000	21	27	16500
2	1	?	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	—	152	mpfi	2.68	3.47	9.0	154	5000	19	26	16500
3	2	164	audi	gas	std	four	sedan	4wd	front	99.8	—	109	mpfi	3.19	3.40	10.0	102	5500	24	30	13950
4	2	164	audi	gas	std	two	sedan	fwd	front	99.8	—	136	mpfi	3.19	3.40	8.0	115	5500	18	22	17450
5	2	?	audi	gas	std	four	sedan	fwd	front	99.8	—	136	mpfi	3.19	3.40	8.5	110	5500	19	25	15250
6	1	158	audi	gas	std	four	sedan	fwd	front	105.8	—	136	mpfi	3.19	3.40	8.5	110	5500	19	25	17710
7	1	?	audi	gas	std	four	wagon	fwd	front	105.8	—	136	mpfi	3.19	3.40	8.5	110	5500	19	25	18920
8	1	158	audi	gas	turbo	four	sedan	fwd	front	105.8	—	131	mpfi	3.13	3.40	8.3	140	5500	17	20	23875
9	0	?	audi	gas	turbo	two	hatchback	4wd	front	99.5	—	131	mpfi	3.13	3.40	7.0	160	5500	16	22	?

ТИПЫ ДАННЫХ

df.dtypes

```
symboling          int64
normalized-losses    object
make                object
fuel-type            object
aspiration           object
num-of-doors         object
body-style            object
drive-wheels          object
engine-location        object
wheel-base           float64
length               float64
width                float64
height               float64
curb-weight           int64
engine-type            object
num-of-cylinders       object
engine-size           int64
fuel-system            object
bore                 object
stroke               object
compression-ratio     float64
horsepower            object
peak-rpm              object
city-mpg              int64
highway-mpg            int64
price                object
dtype: object
```

ПОЯСНЕНИЕ ДАННЫХ

dataframe.describe()

df.describe() # обработает только цифровые типы данных

	symboling	wheel-base	length	width	height	curb-weight	engine-size	compression-ratio	city-mpg	highway-mpg
count	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000
mean	0.834146	98.756585	174.049268	65.907805	53.724878	2555.565854	126.907317	10.142537	25.219512	30.751220
std	1.245307	6.021776	12.337289	2.145204	2.443522	520.680204	41.642693	3.972040	6.542142	6.886443
min	-2.000000	86.600000	141.100000	60.300000	47.800000	1488.000000	61.000000	7.000000	13.000000	16.000000
25%	0.000000	94.500000	166.300000	64.100000	52.000000	2145.000000	97.000000	8.600000	19.000000	25.000000
50%	1.000000	97.000000	173.200000	65.500000	54.100000	2414.000000	120.000000	9.000000	24.000000	30.000000
75%	2.000000	102.400000	183.100000	66.900000	55.500000	2935.000000	141.000000	9.400000	30.000000	34.000000
max	3.000000	120.900000	208.100000	72.300000	59.800000	4066.000000	326.000000	23.000000	49.000000	54.000000

describe all the columns in "df"

df.describe(include = "all") # обработает все

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke	compression-ratio	horsepower	peak-rpm	city-mpg	highway-mpg	price
count	205.000000	205	205	205	205	205	205	205	205.000000	205	205.000000	205	205	205	205.000000	205	205	205.000000	205	205.000000	205
unique	NaN	52	22	2	2	3	5	3	2	NaN	—	NaN	8	39	37	NaN	60	24	NaN	NaN	187
top	NaN	?	toyota	gas	std	four	fwd	front	NaN	—	NaN	mpfi	3.62	3.40	NaN	68	5500	NaN	NaN	?	
freq	NaN	41	32	185	168	114	96	120	202	NaN	—	NaN	94	23	20	NaN	19	37	NaN	NaN	4
mean	0.834146	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	98.756585	—	126.907317	NaN	NaN	NaN	10.142537	NaN	NaN	25.219512	NaN	
std	1.245307	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	6.021776	—	41.642693	NaN	NaN	NaN	3.972040	NaN	NaN	6.542142	6.886443	
min	-2.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	86.600000	—	61.000000	NaN	NaN	NaN	7.000000	NaN	NaN	13.000000	16.000000	
25%	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	94.500000	—	97.000000	NaN	NaN	NaN	8.600000	NaN	NaN	19.000000	25.000000	
50%	1.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	97.000000	—	120.000000	NaN	NaN	NaN	9.000000	NaN	NaN	24.000000	30.000000	
75%	2.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	102.400000	—	141.000000	NaN	NaN	NaN	9.400000	NaN	NaN	30.000000	34.000000	
max	3.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	120.900000	—	326.000000	NaN	NaN	NaN	23.000000	NaN	NaN	49.000000	54.000000	

Теперь мы можем увидеть, сколько здесь уникальных значений, какое из них является верхним значением и частоту верхних значений в столбцах с объектной типизацией.

Некоторые значения в таблице выше отображаются как "NaN". Это потому, что эти значения недоступны для определенного типа столбцов.

ВОПРОС #3:

Вы можете выбрать столбцы фрейма данных, указав имя каждого столбца.

Например, вы можете выбрать три столбца следующим образом:

dataframe[['столбец 1', 'столбец 2', 'столбец 3']].

Где "column" - это имя столбца, вы можете применить метод ".describe()" для получения статистики этих столбцов следующим образом:

dataframe[[' колонка 1 ', 'колонка 2', 'колонка 3']].describe()

Примените метод ".describe()" к столбцам 'length' и 'compression-ratio'.

df[['length', 'compression-ratio']].describe()

	length	compression-ratio
count	205.000000	205.000000
mean	174.049268	10.142537
std	12.337289	3.972040
min	141.100000	7.000000
25%	166.300000	8.600000
50%	173.200000	9.000000
75%	183.100000	9.400000
max	208.100000	23.000000

dataframe.info()

Этот метод печатает информацию о DataFrame, включая тип индекса dtype и столбцы, значения non-null и использование памяти.

df.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 26 columns):
 #   Column           Non-Null Count  Dtype  
 --- 
 0   symboling        205 non-null    int64  
 1   normalized-losses 205 non-null    object  
 2   make              205 non-null    object  
 3   fuel-type         205 non-null    object  
 4   aspiration        205 non-null    object  
 5   num-of-doors      205 non-null    object  
 6   body-style        205 non-null    object  
 7   drive-wheels      205 non-null    object  
 8   engine-location   205 non-null    object  
 9   wheel-base        205 non-null    float64 
 10  length            205 non-null    float64 
 11  width             205 non-null    float64 
 12  height            205 non-null    float64 
 13  curb-weight       205 non-null    int64  
 14  engine-type       205 non-null    object  
 15  num-of-cylinders  205 non-null    object  
 16  engine-size       205 non-null    int64  
 17  fuel-system       205 non-null    object  
 18  bore              205 non-null    object  
 19  stroke            205 non-null    object  
 20  compression-ratio 205 non-null    float64 
 21  horsepower         205 non-null    object  
 22  peak-rpm          205 non-null    object  
 23  city-mpg          205 non-null    int64  
 24  highway-mpg        205 non-null    int64  
 25  price              205 non-null    object  
dtypes: float64(5), int64(5), object(16)
memory usage: 28.9+ KB
```

Pre-processing Data in Python

предварительная обработка данных - это необходимый шаг в анализе данных.

Это процесс преобразования или отображения данных из одной необработанной формы в другой формат, чтобы сделать их готовыми к дальнейшему анализу.

Предварительную обработку данных часто называют **очисткой данных или обработкой данных**, и, вероятно, существуют и другие термины.

Вот темы, которые мы будем рассмотрим в этом модуле.

- Во-первых, мы покажем вам, как определять и обрабатывать отсутствующие значения.

Состояние пропущенного значения возникает всякий раз, когда запись данных остается пустой.

- Затем мы рассмотрим форматы данных. Данные из разных источников могут быть в разных форматах, в различных единицах измерения или в различных соглашениях.
- Мы познакомимся с некоторыми методами в Python Pandas, которые могут стандартизировать значения в один и тот же формат, или единицу измерения, или соглашение.
- После этого мы рассмотрим нормализацию данных.

Learning Objectives

- Identify and handle missing values
- Data Formatting
- Data Normalization (centering /scaling)
- Data Binning
- Turning Categorical values to numeric variables

-

Различные столбцы числовых данных могут иметь очень разные диапазоны и прямое сравнение часто не имеет смысла.

Нормализация - это способ привести все данные в схожий диапазон для более полезного сравнения.

В частности, мы сосредоточимся на технике центрирования и масштабирования.

Затем мы познакомимся с биннингом данных.

Биннинг создает более крупные категории из набора числовых значений.

Это особенно полезно для сравнения между группами данных.

Наконец, мы поговорим о категориальных переменных и покажем вам, как преобразовать

категориальные значения в числовые переменные чтобы облегчить статистическое моделирование.

В Python мы обычно выполняем операции вдоль столбцов Каждая строка столбца представляет собой выборку, т.е. различные подержанные автомобили в базе данных.

Вы получаете доступ к столбцу, указывая его имя.

Например, вы можете получить доступ к символам и стилю тела.

Simple Dataframe Operations

The screenshot shows a Jupyter Notebook interface. At the top, there are two code cells:

```
df["symboling"]
```

```
df["body-style"]
```

Below the code, two DataFrames are displayed side-by-side. The first DataFrame has columns: symboling, normalized-losses, make, fuel-type, aspiration, num-of-doors, body-style, drive-wheels, engine-location, wheel-base, engine-size, fuel-system, bore, stroke, compression-ratio. The second DataFrame has columns: symboling, normalized-losses, make, fuel-type, aspiration, num-of-doors, body-style, drive-wheels, engine-location, wheel-base, engine-size, fuel-system, bore, stroke, compression-ratio. Both DataFrames have 9 rows of data. Red arrows point from the column names in the code cells down to the corresponding columns in the DataFrames.

		symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	engine-size	fuel-system	bore	stroke	compression-ratio
0	5	?	oldsmobile	gas	std	two	convertible	rwd	front	98.6	...	130	mpfi	9.47	2.68	9.0
1	3	?	oldsmobile	gas	std	two	convertible	rwd	front	98.6	...	130	mpfi	9.47	2.68	9.0
2	1	?	oldsmobile	gas	std	two	notchback	rwd	front	94.5	...	152	mpfi	9.68	3.47	9.0
3	2	184	audi	gas	std	four	sedan	rwd	front	99.8	...	109	mpfi	3.19	3.40	10.0
4	2	184	audi	gas	std	four	sedan	4wd	front	99.4	...	138	mpfi	3.19	3.40	8.0
5	2	?	audi	gas	std	two	sedan	rwd	front	99.8	...	138	mpfi	3.19	3.40	8.5
6	1	188	audi	gas	std	four	sedan	rwd	front	103.8	...	136	mpfi	3.18	3.40	8.5
7	1	?	audi	gas	std	four	wagon	rwd	front	103.8	...	136	mpfi	3.18	3.40	8.5
8	1	188	audi	gas	turbo	four	sedan	rwd	front	103.8	...	131	mpfi	3.18	3.40	8.3
9	0	?	audi	gas	turbo	two	hatchback	4wd	front	99.5	...	131	mpfi	3.18	3.40	7.0

Каждый из этих столбцов представляет собой серию Panda. В Python существует множество способов манипулирования Dataframes.

Например, вы можете добавить значение к каждой записи в столбце.

Чтобы добавить одно значение к каждой записи символа, используйте эту команду.

Это изменит каждое значение в столбца рамки данных, добавляя единицу к текущему значению.

How to deal with missing data?

Check with the data collection source

Drop the missing values

- drop the variable
- drop the data entry

Replace the missing values

- replace it with an average (of similar datapoints)
- replace it by frequency
- replace it based on other functions

В этом видеоролике мы познакомимся с распространенной проблемой недостающих значений, а также стратегии что делать, когда вы сталкиваетесь с отсутствующими значениями в ваших данных.

Когда для конкретного наблюдения не сохранилось ни одного значения данных, мы говорим, что эта характеристика имеет пропущенное значение.

Обычно пропущенное значение в наборе данных выглядит как вопросительный знак и ноль или просто пустая ячейка.

Missing Values

- What is missing value?
- Missing values occur when no data value is stored for a variable (feature) in an observation.
- Could be represented as "?", "N/A", 0 or just a blank cell.

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location
0	3	NaN	alfa-romero	gas	std	two	convertible	rwd	front

В приведенном здесь примере нормализованная характеристика потерь имеет пропущенное значение, которое представлено в виде NaN.

Но как можно справиться с отсутствующими данными?

Тем не менее, вот типичные варианты, которые вы можете рассмотреть.

- Первый - проверить, может ли человек или группа, которая собирала данные, может вернуться назад и выяснить, каким должно быть фактическое значение.
- Другая возможность - просто удалить данные, в которых обнаружено недостающее значение.

Когда вы удаляете данные, вы можете удалить либо всю переменную, либо только одну запись данных с отсутствующим значением.

- Если у вас не так много наблюдений с отсутствующими данными, обычно лучше всего отбросить конкретную запись.

Если вы удаляете данные, вы хотите сделать что-то, что окажет наименьшее влияние.

Замена данных лучше, так как данные не пропадают зря. Однако она менее точна, поскольку нам необходимо заменить отсутствующие данные предположением о том, какими должны быть данные.

- Одним из стандартных методов размещения данных является замена недостающие значения средним значением всей переменной.

В качестве примера, предположим, что у нас есть несколько записей с отсутствующими значениями для столбца "Нормализованные потери", а среднее значение столбца для записей с данными равно 4500.

Хотя у нас нет возможности получить точное предположение о том, каково должно быть недостающее значение в колонке нормализованных потерь, можно приблизительно определить их значения, используя среднее значение столбца 4500.

Но что если значения нельзя усреднить, как в случае с категориальными переменными?

Для такой переменной, как тип топлива, не существует среднего значения типа топлива, поскольку значения переменной не являются числами.

В этом случае можно попробовать использовать режим, наиболее распространенный, например, бензин.

Наконец, иногда мы можем найти другой способ угадать недостающие данные.

Обычно это происходит потому, что собранные данные знает что-то дополнительное об отсутствующих данных.

Например, он может знать, что недостающими значениями, как правило, являются

старые автомобили, а нормированные потери старых автомобилей значительно выше, чем у среднего автомобиля.

И, конечно, наконец, в некоторых случаях вы можете просто оставить отсутствующие данные как отсутствующие данные.

По тем или иным причинам, может быть полезно сохранить это наблюдение, даже если некоторые характеристики отсутствуют.

Теперь давайте разберемся, как отбросить отсутствующие значения или заменить отсутствующие значения в Python.

Чтобы удалить данные, содержащие отсутствующие значения

В библиотеке Panda есть встроенный метод под названием dropna.

По сути, с помощью метода dropna, вы можете выбрать отбрасывание строк или столбцов, которые содержат отсутствующие значения, например NaN.

Поэтому вам нужно указать доступ, равный нулю, чтобы отбросить строки или доступ, равный единице, чтобы отбросить столбцы, содержащие пропущенные значения.

В данном примере в столбце "Цена" отсутствует значение.

Поскольку цена подержанных автомобилей - это то, что мы пытаемся предсказать в нашем предстоящем анализе, мы должны удалить автомобили, строки, в которых не указана цена.

Это можно сделать одной строкой кода с помощью **dataframe.dropna**.

How to drop missing values in Python

- Use `dataframes.dropna ()` :

highway-mpg	price
...	...
20	23875
22	NaN
29	16430
...	...

→

highway-mpg	price
...	...
20	23875
29	16430
...	...

axis=0 drops the entire row
axis=1 drops the entire column

```
df.dropna(subset=["price"], axis=0, inplace = True)  
df = df.dropna(subset=["price"], axis=0)
```

Subset – подмножество данных

Установка аргумента `inplace` в `true`, позволяет выполнить модификацию непосредственно в наборе данных.

In place = true, просто записывает результат обратно в кадр данных.

Это эквивалентно этой строке кода. Не забывайте, что эта строка кода не изменяет фрейм данных, но это хороший способ убедиться, что вы выполняете правильную операцию.

Чтобы изменить фрейм данных, вы должны установить параметр **inplace = true**.

Вы всегда должны обращаться к документации, если если вы не знакомы с функцией или методом.

Чтобы заменить отсутствующие значения, например, NaNs, на действительные значения, в библиотеке Pandas есть встроенный **метод replace**, который можно использовать для заполнения недостающих значений новыми вычисленными значениями.

В качестве примера, предположим, что мы хотим заменить отсутствующие значения переменной переменной нормализованные потери средним значением переменной.

How to replace missing values in Python

Use `dataframe.replace(missing_value, new_value)`:

The diagram illustrates the process of replacing a missing value in a DataFrame. On the left, there is a table with two columns: 'normalized-losses' and 'make'. The 'normalized-losses' column contains values like 164, 164, NaN, 158, and ...; the 'make' column contains 'audi' repeated. An arrow points from this state to the right. On the right, the same table is shown, but the NaN value in the 'normalized-losses' column has been replaced by the mean value, which is 162. The 'make' column remains unchanged.

normalized-losses	make
...	...
164	audi
164	audi
NaN	audi
158	audi
...	...

→

normalized-losses	make
...	...
164	audi
164	audi
162	audi
158	audi
...	...

```
mean = df["normalized-losses"].mean()  
df["normalized-losses"].replace(np.nan, mean)
```

Таким образом, недостающее значение должно быть заменено средним значением записей в этом столбце.

В Python мы сначала вычисляем среднее значение столбца. Затем мы используем метод `replace`, чтобы указать в качестве первого параметра значение, которое мы хотим заменить, в данном случае `NaN`.

Вторым параметром является значение, которое мы хотели бы, то есть среднее значение в данном примере. Это довольно упрощенный способ замены отсутствующих значений. **Конечно, существуют и другие методы, например, замена недостающих значений на среднее значение группы, а не всего набора данных.**

Но не забывайте о других способах борьбы с отсутствующими данными.

Вы всегда можете проверить набор или источник данных более высокого качества или в некоторых случаях вы можете оставить отсутствующие данные как отсутствующие данные.

ФОРМАТИРОВАНИЕ ДАННЫХ

В этом видео мы рассмотрим проблему данных с различными форматами, единицы измерения и условные обозначения, а также методы pandas, которые помогают нам справиться с этими проблемами.

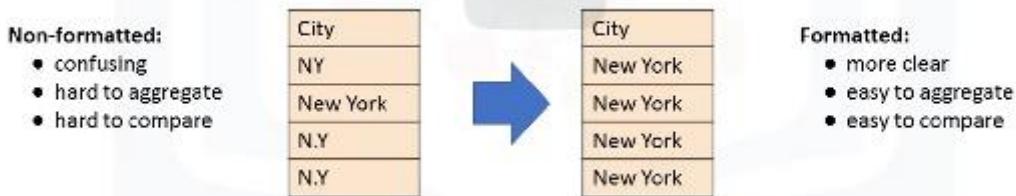
Данные обычно собираются из разных мест разными людьми разными людьми, которые могут храниться в разных форматах.

Форматирование данных означает приведение данных к общему стандарту выражения, который позволяет пользователям проводить осмысленные сравнения.

Как часть очистки набора данных, форматирование данных обеспечивает их согласованность и понятность. Например, люди могут использовать различные выражения для обозначения Нью-Йорка, например, верхний регистр N верхний регистр Y, верхний регистр N нижний регистр Y, верхний регистр N верхний регистр Y и Нью-Йорк. Иногда такие нечистые данные полезны.

Data Formatting

- Data are usually collected from different places and stored in different formats.
- Bringing data into a common standard of expression allows users to make meaningful comparison.



Например, если вы изучаете различные варианты написания слова "Нью-Йорк", то это именно те данные, которые вам нужны.

Или если вы ищете способы выявления мошенничества, возможно, написание N.Y. с большей вероятностью предсказать аномалию, чем если бы кто-то написал Нью-Йорк полностью.

Но, возможно, чаще всего мы просто хотим рассматривать их все как одну и ту же сущность или формат, чтобы облегчить статистический анализ в дальнейшем.

Обращаясь к нашему набору данных о подержанных автомобилях, в наборе данных есть функция под названием city-miles per gallon, которая обозначает расход топлива автомобиля в милях на галлон.

Однако вы можете быть человеком, живущим в стране, где используются метрические единицы.

Поэтому вы захотите преобразовать эти значения в литры на 100 километров, метрическую версию. Чтобы преобразовать мили на галлон в литры на 100 км, нужно разделить 235 на каждое значение в столбце "город-мили на галлон".

В Python это можно легко сделать одной строкой кода.

Вы берете столбец, устанавливаете его равным 235, делите его на весь столбец. Во второй строке кода, переименуйте название столбца с city-miles per gallon на city-liters per 100 kilometers, используя метод переименования фрейма данных.

Applying calculations to an entire column

- Convert "mpg" to "L/100km" in Car dataset.

The diagram illustrates a data transformation process. On the left, there is a table labeled 'city-mpg' containing five rows of data: 21, 21, 19, and two ellipsis ('...'). An arrow points from this table to another table on the right, which is labeled 'city-L/100km'. This second table also contains five rows of data: 11.2, 11.2, 12.4, and two ellipsis ('...').

city-mpg
21
21
19
...

city-L/100km
11.2
11.2
12.4
...

```
df["city-mpg"] = 235/df["city-mpg"]

df.rename(columns={"city_mpg": "city-L/100km"}, inplace=True)
```

По ряду причин, в том числе при импорте набора данных в Python, тип данных может быть установлен неверно.

Например, здесь мы заметили, что тип данных, присвоенный функции price, - object.

Incorrect data types

- Sometimes the wrong data type is assigned to a feature.

```
df["price"].tail(5)

 200    16845
 201    19045
 202    21485
 203    22470
 204    22625
Name: price, dtype: object
```

Хотя на самом деле ожидаемый тип данных должен быть целым числом или типом float.

Для последующего анализа важно изучить тип данных характеристики и преобразовать их в правильные типы данных.

В противном случае разработанные модели в дальнейшем могут вести себя странно, а совершенно верные данные могут быть расценены как отсутствующие.

В pandas существует множество типов данных.

Объекты могут быть буквами или словами.

Int64 - это целые числа, а floats - вещественные числа.

Существует множество других типов, которые мы не будем обсуждать.

Чтобы определить тип данных, в Python мы можем использовать метод **dataframe.dtypes ()** и проверить тип данных каждой переменной в кадре данных.

В случае неправильных типов данных, метод **dataframe.astype ()** может быть использован для преобразования типа данных из одного формата в другой.

Correcting data types

To identify data types:

- Use **dataframe.dtypes ()** to identify data type.

To convert data types:

- Use **dataframe.astype ()** to convert data type.

Example: convert data type to integer in column "price"

```
df["price"] = df["price"].astype("int")
```

Например, используя `astype int` для столбца `price`, вы можете преобразовать столбец объекта в переменную целочисленного типа.

Data Normalization in Python

Data Normalization

- Uniform the features value with different range.

length	width	height
168.8	64.1	48.8
168.8	64.1	48.8
171.2	65.5	52.4
176.6	66.2	54.3
176.6	66.4	54.3
177.3	66.3	53.1
192.7	71.4	55.7
192.7	71.4	55.7
192.7	71.4	55.9

scale	[150,250]	[50,100]	[50,100]
impact	large	small	small

Если мы посмотрим на набор данных о подержанных автомобилях, мы заметим, что длина признаков колеблется в пределах 150-250, а ширина и высота - от 50 до 100.

Мы можем захотеть нормализовать эти переменные, чтобы диапазон значений был согласованным. Такая нормализация может облегчить некоторые статистические анализы в дальнейшем. Обеспечивая согласованность диапазонов

между переменными, нормализация позволяет справедливо сравнивать различные характеристики, убедиться, что они оказывают одинаковое влияние.

Это также важно для вычислительных целей. Вот еще один пример, который поможет вам понять, почему нормализация важна.

Рассмотрим набор данных, содержащий две характеристики - возраст и доход. Возраст варьируется от 0 до 100 лет, а доход варьируется от 0-20 000 и выше.

Доход примерно в 1 000 раз больше, чем возраст, и колеблется в пределах 20 000-500 000.

Таким образом, эти две характеристики находятся в очень разных диапазонах.

Когда мы проводим дальнейший анализ, например, линейную регрессию, атрибут дохода будет по своей природе окажет большее влияние на результат из-за своего большего значения.

Но это не обязательно означает, что он более важен как предиктор.

Таким образом, характер данных смещает линейную регрессионную модель к более высокому значению дохода, чем возраста.

Чтобы избежать этого, мы можем нормализовать эти две переменные в значения, которые варьируются от нуля до единицы.

Data Normalization

age	income
20	100000
30	20000
40	500000



age	income
0.2	0.2
0.3	0.04
0.4	1

Not-normalized

- "age" and "income" are in different range.
- hard to compare
- "income" will influence the result more

Normalized

- similar value range.
- similar intrinsic influence on analytical model.

На Максимальное значение столбца разделили все значения столбца для приведения к одному диапазону:

$$100000 : 500000 = 0.2 \text{ -- доход}$$

$$20 : 100 = 0.2 \text{ - возраст}$$

Сравните две таблицы справа. После нормализации обе переменные теперь имеют одинаковое влияние на модели, которые мы построим позже.

Существует несколько способов нормализации данных.

Я остановлюсь на трех методах.

- Первый метод, называемый простым масштабированием признаков, просто делит каждое значение на максимальное значение для данного признака. В результате новые значения оказываются в диапазоне от нуля до единицы.
- Второй метод, называемый min-max, берет каждое значение X_{old} вычитается из минимального значения этой характеристики, затем делится на диапазон этого признака. Опять же, полученные новые значения находятся в диапазоне от нуля до единицы.
- Третий метод называется z-score или стандартный балл. В этой формуле для каждого значения вы вычитаете **mu, которое является средним значением признака**, а затем делите на стандартное отклонение сигма.

Methods of normalizing data

Several approaches for normalization:

①

$$x_{new} = \frac{x_{old}}{x_{max}}$$

Simple Feature scaling

②

$$x_{new} = \frac{x_{old} - x_{min}}{x_{max} - x_{min}}$$

Min-Max

③

$$x_{new} = \frac{x_{old} - \mu}{\sigma}$$

Z-score

Полученные значения крутятся вокруг нуля, и обычно колеблются между отрицательными тремя и положительными тремя, но могут быть выше или ниже.

Следуя нашему предыдущему примеру, мы можем применить метод нормализации к характеристике длины. Сначала мы используем простой метод масштабирования признака, где мы делим его на максимальное значение в

характеристике.

Min-max in Python

With Pandas:

length	width	height
168.8	64.1	48.8
168.8	64.1	48.8
180.0	65.5	52.4
...

→

length	width	height
0.41	64.1	48.8
0.41	64.1	48.8
0.58	65.5	52.4
...

```
df["length"] = (df["length"] - df["length"].min()) /  
                (df["length"].max() - df["length"].min())
```

Используя метод **pandas max**, это можно сделать всего за одну строку кода. А вот метод min-max для характеристики длины. Мы вычитаем каждое значение из минимума этого столбца, затем делим его на диапазон этого столбца.

Максимум минус минимум.

Наконец, мы применяем метод z-score для характеристики длины, чтобы нормализовать значения. Здесь мы применим метод среднего и STD к признаку длины. Метод Mean вернет среднее значение признака в наборе данных, а метод STD вернет стандартное отклонение признака в наборе данных.

БИННИНГ

В этом видео мы поговорим о бинировании как методе предварительной обработки данных.

Биннинг - это когда вы группируете значения в бины. Например, вы можете разделить значение "возраст" на [0 - 5], [6 - 10], [11 - 15] и так далее.

Иногда бинирование может повысить точность прогностических моделей.

Кроме того, иногда мы используем биннинг данных, чтобы сгруппировать набор числовых значений в меньшее число бинов, чтобы лучше понять распределение данных.

Например, "цена" здесь - это атрибут, диапазон которого составляет от 5 000 до 45 500.

Binning

- Binning: Grouping of values into "bins"
- Converts numeric into categorical variables
- Group a set of numerical values into a set of "bins"
- "price" is a feature range from 5,000 to 45,500
(in order to have a better representation of price)

price: 5000, 10000, 12000, 12000, 30000, 31000, 39000, 44000, 44500



Используя биннинг, мы разделим цену на три категории: низкая цена, средняя цена и высокая цена.

В реальном наборе данных автомобилей "цена" - это числовая переменная в диапазоне от 5188 до 45400, она имеет 201 уникальное значение.

Мы можем разделить их на 3 бина: автомобили с низкой, средней и высокой ценой.

Binning in Python pandas

price
13495
16500
18920
41315
5151
6295
...



price	price-binned
13495	Low
16500	Low
18920	Medium
41315	High
5151	Low
6295	Low
...	...

В Python мы можем легко реализовать биннинг: Мы хотим получить 3 бина с одинаковой шириной бина, поэтому нам нужны 4 числа в качестве делителей, которые находятся на одинаковом расстоянии друг от друга.

Сначала мы используем функцию numpy "linspace", чтобы вернуть массив "bins", который содержит 4 равноотстоящих друг от друга числа на заданном интервале цены.

Мы создаем список "group_names", который содержит различные имена бинов. Мы используем функцию pandas "cut" для сегментации и сортировки значений данных по бинам.

Binning in Python pandas

price
13495
16500
18920
41315
5151
6295
...



price	price-binned
13495	Low
16500	Low
18920	Medium
41315	High
5151	Low
6295	Low
...	...

```
bins = np.linspace(min(df["price"]), max(df["price"]), 4)  
group_names = ["Low", "Medium", "High"]  
  
df[“price-binned”] = pd.cut(df[“price”], bins, labels=group_names, include_lowest=True )
```

Затем можно использовать гистограммы для визуализации распределения данных после того, как они были разделены на бины.

Это гистограмма, которую мы построили на основе разбиения на бины, которое мы применили в функции price функции.

Из графика видно, что большинство автомобилей имеют низкую цену, и только очень немногие автомобили имеют высокую цену.

ПЕРЕВОД ПЕРЕМЕННЫХ СТРОКОВЫХ В ЧИСЛА

Categorical → Numeric

Solution:

- Add dummy variables for each unique category
- Assign 0 or 1 in each category

Car	Fuel	...	gas	diesel
A	gas	...	1	0
B	diesel	...	0	1
C	gas	...	1	0
D	gas	...	1	0

“One-hot encoding”

В этом видео мы обсудим, как превратить категориальные переменные в количественные переменные в Python. **Большинство статистических моделей не могут принимать на вход объекты или строки** и для обучения модели в качестве входных данных принимают только числа.

В наборе данных об автомобилях признак типа топлива как категориальная переменная имеет два значения, газ или дизель, которые имеют строковый формат.

Для дальнейшего анализа Джерри должен преобразовать эти переменные в какой-либо числовой формат.

Мы кодируем значения, добавляя новые признаки, соответствующие каждому уникальному элементу исходной характеристики, которую мы хотим закодировать.

В случае, когда признак "топливо" имеет два уникальных значения, газ и дизель, мы создаем два новых признака - газ и дизель. Когда значение встречается в исходном признаке, мы устанавливаем соответствующее значение в единицу в новой функции.

Остальные признаки имеют нулевое значение. В примере с топливом для автомобиля B значение топлива - дизельное.

Поэтому мы устанавливаем признак diesel равным единице, а признак gas - нулю. Аналогично, для автомобиля D, значение топлива - газ.

Поэтому мы устанавливаем признак gas равным единице, а признак diesel равным нулю.

Эту технику часто называют "кодированием по принципу "один к одному".

В Pandas мы можем использовать метод **get_dummies** для преобразования категориальных переменных в фиктивные переменные.

В Python преобразование категориальных переменных в фиктивные переменные очень просто.

Следуя примеру, метод pd.get_dummies получает столбец типа топлива и создает рамку данных dummy_variable_1. Метод get_dummies автоматически генерирует список чисел, каждое из которых соответствует определенному разряду переменной.

Dummy variables in Python pandas

- Use `pandas.get_dummies()` method.
- Convert categorical variables to dummy variables (0 or 1)

fuel	gas	diesel
gas	1	0
diesel	0	1
gas	1	0
gas	1	0

```
pd.get_dummies(df['fuel'])
```

ИТОГО

Определять и обрабатывать пропущенные значения: Отбрасывать строки с неполной информацией и вписывать недостающие данные, используя средние значения.

Понимать форматирование данных: Упорядочивать характеристики набора данных и делать их значимыми для анализа данных.

Применять нормализацию к набору данных: Понимание целесообразности использования масштабирования признаков для ваших данных и того, как нормализация и стандартизация оказывают различное влияние на анализ данных.

РАБОТА С НЕДОСТАЮЩИМИ ДАННЫМИ, ОЧИСТКА

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-s...
0	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	
1	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	
2	1	?	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	
3	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	
4	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	

В наборе данных по автомобилям отсутствующие данные сопровождаются вопросительным знаком "?". Мы заменяем "?" на `NaN` (*Not a Number*), маркер отсутствующих значений по умолчанию в Python, по причинам скорости вычислений и удобства. Здесь мы используем функцию:

`df.replace(A, B, inplace = True)`

для замены *A* на *B*.

```
import numpy as np
```

```
# replace "?" to NaN
```

```
df.replace("?", np.nan, inplace = True) # "?" – меняем этот символ на np.nan (NOT A NUMBER из библиотеки Numpy)
```

```
df.head(5)
```

:	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...
0	3	NaN	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...
1	3	NaN	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...
2	1	NaN	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...
3	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...
4	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...

Отсутствующие значения преобразуются по умолчанию. Мы используем следующие функции для определения этих пропущенных значений. Существует два метода обнаружения недостающих данных:

.isnull()

.notnull()

```
missing_data = df.isnull() # где missing_data это переменная.
```

```
missing_data.head(5)
```

```
# Результатом является логическое значение, указывающее, действительно ли значение, переданное в аргумент, является отсутствующими данными
```

Используя **цикл for** в Python, мы можем быстро определить количество отсутствующих значений в каждом столбце. Как упоминалось выше, "True" означает отсутствующее значение, а "False" означает, что значение присутствует в наборе данных. В теле цикла for метод ".value_counts()" подсчитывает количество значений "True".

```
for column in missing_data.columns.values.tolist():
```

```
    print(column)
```

```
    print (missing_data[column].value_counts())
```

```
    print("")
```

```
symboling
False    205
Name: symboling, dtype: int64

normalized-losses
False    164
True     41
Name: normalized-losses, dtype: int64

make
False    205
Name: make, dtype: int64

fuel-type
False    205
Name: fuel-type, dtype: int64

aspiration
False    205
Name: aspiration, dtype: int64

num-of-doors
False    203
True     2
Name: num-of-doors, dtype: int64

body-style
False    205
Name: body-style, dtype: int64

drive-wheels
False    205
Name: drive-wheels, dtype: int64
```

Исходя из приведенной выше сводки, каждый столбец содержит 205 строк данных, а семь столбцов содержат отсутствующие данные:

- "normalized-losses": 41 отсутствующие данные
- "num-of-doors": 2 отсутствующие данные
- " bore ": 4 отсутствующие данные
- " stroke ": 4 отсутствующих данных
- " horsepower ": 2 отсутствующие данные
- "пиковые обороты": 2 отсутствующие данные
- "цена": 4 отсутствующие данные

Как работать с отсутствующими данными?

Отбросьте данные

- а. Отбросьте весь ряд
- б. Опустить весь столбец

Заменить данные

- а. Заменить на среднее значение
- б. Заменить на частоту
- с. Заменить на другие функции

Замените на часто встречающееся значение :

- "num-of-doors": 2 отсутствующих данных, замените их на "4".

Причина: 84% седанов - четырехдверные. Так как четыре двери встречаются чаще всего, то вероятность их появления наиболее высока.

Отбросьте весь ряд:

- "цена": 4 недостающих данных, просто удалите весь ряд

Причина: цена - это то, что мы хотим предсказать. Любая запись данных без данных о цене не может быть использована для прогнозирования; поэтому любая строка без данных о цене не является полезной для нас

- **Рассчитайте среднее значение для столбца «нормализованные потери»**

`avg_norm_loss = df["normalized-losses"].astype("float").mean(axis=0) # где avg_norm_loss – переменная, ["имя столбца"] .astype – метод для стандартизации типа данных («float») – тип данных в котором будет вестись расчет, .mean – метод для расчета среднего арифметического в массиве (axis = 0) – нулевая ось`

```
print("Average of normalized-losses:", avg_norm_loss)
```

Замените «NaN» средним значением в столбце «нормализованные потери».

```
df["normalized-losses"].replace(np.nan, avg_norm_loss, inplace=True)
```

Calculate the MEAN VALUE for the "BORE" COLUMN

```
avg_bore = df['bore'].astype('float').mean(axis=0)
```

```
print("Average of bore:", avg_bore)
```

ЗАМЕНИТЕ «NAN» СРЕДНИМ ЗНАЧЕНИЕМ В СТОЛБЦЕ «bore».

```
df["bore"].replace(np.nan, avg_bore, inplace=True)
```

ВОПРОС 1: НА ОСНОВЕ ПРИВЕДЕННОГО ВЫШЕ ПРИМЕРА ЗАМЕНИТЕ NAN В СТОЛБЦЕ «STROKE» НА СРЕДНЕЕ ЗНАЧЕНИЕ.

```
#Calculate the mean value for "stroke" column
```

```
avg_stroke = df["stroke"].astype("float").mean(axis = 0)
```

```
print("Average of stroke:", avg_stroke)

# replace NaN by mean value in "stroke" column
df["stroke"].replace(np.nan, avg_stroke, inplace = True)
```

OUTPUT: Average value stroke 3.255422885572139

ВЫЧИСЛИТЬ СРЕДНЕЕ ЗНАЧЕНИЕ ДЛЯ СТОЛБЦА «PEAK-RPM»

```
avg_peakrpm=df['peak-rpm'].astype('float').mean(axis=0)
```

**переменная = датафрейм [‘имя столбца’] .методПреобразованияЧисел
(‘Тип_данных для преобразования’).МетодРасчетаСреднегоЗначения (ось=0
или 1)**

```
print("Average peak rpm:", avg_peakrpm)
```

**Заменить значение NOT A NUMBER , типа данных из библиотеки NumPy на
среднее значение по столбцу (avg_peakrpm)**

```
df['peak-rpm'].replace(np.nan, avg_peakrpm, inplace=True)
```

Чтобы посмотреть, какие значения присутствуют в определенном столбце, мы можем использовать метод ".value_counts()":

```
df['peak-rpm'].value_counts().head(5) # ограничила 5 записями
```

```
5500    37
4800    36
5000    27
5200    23
5400    13
Name: peak-rpm, dtype: int64
```

Мы видим, что четыре двери являются наиболее распространенным типом. Мы также можем использовать метод ".idxmax()" для автоматического вычисления наиболее распространенного типа:

```
df['peak-rpm'].value_counts().idxmax()
```

```
# simply drop whole row with NaN in "price" column
```

```
df.dropna(subset=["price"], axis=0, inplace=True)
```

```
# reset index, because we droped two rows
```

```
df.reset_index(drop=True, inplace=True)
```

ПРОВЕРИМ ТАБЛИЦУ

```
df.head()
```

РАБОТА С НЕЦИФРОВЫМ ТИПОМ ДАННЫХ

```
df["num-of-doors"].replace(np.nan, avg_peakrpm, inplace=True)
```

```
df["num-of-doors"].value_counts() # посчитаем наиболее распространенные типы данных
```

или так

```
df['num-of-doors'].value_counts().idxmax() # наиболее распространенное значение
```

```
df["num-of-doors"].replace(np.nan, "four", inplace=True) # променяем пустые значения на наиболее распространенное ('four')
```

УДАЛИМ ВСЕ СТРОКИ С НЕДОСТАТКОМ ДАННЫХ ПО ЦЕНАМ

```
# simply drop whole row with NaN in "price" column
```

```
df.dropna(subset=["price"], axis=0, inplace=True)
```

```
# reset index, because we droped two rows
```

```
df.reset_index(drop=True, inplace=True)
```

Проверим таблицу

```
df.head()
```

ПРАВИЛЬНЫЙ ФОРМАТ ДАННЫХ

Последний шаг в очистке данных - проверка и убеждение в том, что все данные имеют правильный формат (int, float, text или другой).

- `.dtype()` для проверки типа данных

- `.astype()` для изменения типа данных

Давайте перечислим типы данных для каждого столбца

`df.dtypes`

```
symboling      int64
normalized-losses   object
make          object
fuel-type     object
aspiration    object
num-of-doors   object
body-style    object
drive-wheels   object
engine-location  object
wheel-base    float64
length        float64
width         float64
height        float64
curb-weight   int64
engine-type   object
num-of-cylinders  object
engine-size    int64
fuel-system   object
bore          object
stroke        object
compression-ratio float64
horsepower    object
peak-rpm       object
city-mpg      int64
highway-mpg   int64
price         object
dtype: object
```

ВЫВОД: Как мы видим выше, некоторые столбцы имеют неправильный тип данных. Числовые переменные должны иметь тип '**float**' или '**int**', а переменные со строками, такие как категории, должны иметь тип '**object**'. Например, переменные '**bore**' и '**stroke**' - это числовые значения, описывающие двигатели, поэтому мы должны ожидать, что они будут иметь тип '**float**' или '**int**'; однако они показаны как тип '**object**'.

РЕШЕНИЕ

Мы должны **преобразовать типы данных** в соответствующий формат для каждого столбца **с помощью метода "astype()"**

`df[["bore", "stroke"]] = df[["bore", "stroke"]].astype("float")`

`df[["normalized-losses"]] = df[["normalized-losses"]].astype("int")`

`df[["price"]] = df[["price"]].astype("float")`

`df[["peak-rpm"]] = df[["peak-rpm"]].astype("float")`

ПРОВЕРИМ ПРЕОБРАЗОВАНИЕ

```
df.dtypes
```

```
symboling          int64
normalized-losses   int32
make                object
fuel-type           object
aspiration          object
num-of-doors        object
body-style          object
drive-wheels        object
engine-location     object
wheel-base          float64
length              float64
width               float64
height              float64
curb-weight         int64
engine-type         object
num-of-cylinders    object
engine-size          int64
fuel-system          object
bore                float64
stroke              float64
compression-ratio   float64
horsepower          object
peak-rpm             float64
city-mpg             int64
highway-mpg          int64
price                float64
dtype: object
```

ЕСЛИ КОРОЧЕ, проверим только измененные столбцы

```
df.dtypes[["bore", "stroke", "normalized-losses", "price", "peak-rpm"]]
```

```
bore              float64
stroke            float64
normalized-losses   int32
price              float64
peak-rpm           float64
dtype: object
```

СРАВНИМ

Before

```
symboling          int64
normalized-losses object
make              object
fuel-type         object
aspiration        object
num-of-doors      object
body-style        object
drive-wheels      object
engine-location   object
wheel-base        float64
length            float64
width             float64
height            float64
curb-weight       int64
engine-type       object
num-of-cylinders object
engine-size       int64
fuel-system       object
bore              object
stroke            object
compression-ratio float64
horsepower        object
peak-rpm           object
city-mpg          int64
highway-mpg        int64
price              object
dtype: object
```

After

```
symboling          int64
normalized-losses int32
make              object
fuel-type         object
aspiration        object
num-of-doors      object
body-style        object
drive-wheels      object
engine-location   object
wheel-base        float64
length            float64
width             float64
height            float64
curb-weight       int64
engine-type       object
num-of-cylinders object
engine-size       int64
fuel-system       object
bore              float64
stroke            float64
compression-ratio float64
horsepower        object
peak-rpm           float64
city-mpg          int64
highway-mpg        int64
price              float64
dtype: object
```

СТАНДАРТИЗАЦИЯ ДАННЫХ

Стандартизация данных - это также термин для обозначения определенного типа нормализации данных, когда мы вычитаем среднее значение и делим на стандартное отклонение.

Стандартизация - это процесс преобразования данных в общий формат, позволяющий исследователю провести значимое сравнение.

Пример

Преобразуйте mpg в L/100km:

В нашем наборе данных колонки расхода топлива "city-mpg" и "highway-mpg" представлены в единицах измерения mpg (мили на галлон). Предположим, что мы разрабатываем приложение в стране, где принят стандарт расхода топлива в L/100 КМ.

Нам нужно будет применить преобразование данных для преобразования mpg в L/100km.

FORMULA:

$$L/100km = 235 / mpg$$

Проверим таблицу

```
df.head()
```

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke	compression-ratio	horsepower	peak-rpm	city-mpg	highway-mpg	price
0	3	122	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	111	5000.0	21	27	13495.0
1	3	122	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	111	5000.0	21	27	16500.0
2	1	122	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	9.0	154	5000.0	19	26	16500.0
3	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	10.0	102	5500.0	24	30	13950.0
4	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.40	8.0	115	5500.0	18	22	17450.0

```
# Convert mpg to L/100km by mathematical operation (235 divided by mpg)
```

```
df['city-L/100km'] = 235/df["city-mpg"]
```

```
# check your transformed data
```

```
df.head()
```

stroke	compression-ratio	horsepower	peak-rpm	city-mpg	highway-mpg	price	city-L/100km
2.68	9.0	111	5000.0	21	27	13495.0	11.190476
2.68	9.0	111	5000.0	21	27	16500.0	11.190476
3.47	9.0	154	5000.0	19	26	16500.0	12.368421
3.40	10.0	102	5500.0	24	30	13950.0	9.791667
3.40	8.0	115	5500.0	18	22	17450.0	13.055556

Вопрос 2: Согласно приведенному выше примеру, преобразуйте мили на галлон в л/100 км в столбце «шоссе-миль на галлон» и измените имя столбца на «шоссе-л/100 км».

```
# transform mpg to L/100km by mathematical operation (235 divided by mpg)
```

```
df["highway-mpg"] = 235/df["highway-mpg"]
```

```
# rename column name from "highway-mpg" to "highway-L/100km"
```

```
df.rename(columns={'highway-mpg':'highway-L/100km'}, inplace=True)
```

```
# check your transformed data
```

```
df.head()
```

НОРМАЛИЗАЦИЯ ДАННЫХ

Нормализация - это процесс преобразования значений нескольких переменных в одинаковый диапазон. Типичная нормализация включает в себя масштабирование переменной, чтобы среднее значение переменной было равно 0,

масштабирование переменной, чтобы дисперсия была равна 1, или масштабирование переменной, чтобы значения переменной варьировались от 0 до 1.

Пример

Чтобы продемонстрировать нормализацию, допустим, мы хотим масштабировать столбцы "длина", "ширина" и "высота".

Цель: мы хотим нормализовать эти переменные так, чтобы их значения находились в диапазоне от 0 до 1.

Подход: замените исходное значение на **(исходное значение) разделить на (максимальное значение)**

```
# replace (original value) by (original value)/(maximum value)
```

```
df['length'] = df['length']/df['length'].max()
```

```
df['width'] = df['width']/df['width'].max()
```

Вопрос №3: Согласно приведенному выше примеру нормализуйте столбец «высота».

```
df['height'] = df['height']/df['height'].max()
```

```
# show the scaled columns
```

```
df[["length","width","height"]].head()
```

	length	width	height
0	168.8	64.1	0.816054
1	168.8	64.1	0.816054
2	171.2	65.5	0.876254
3	176.6	66.2	0.908027
4	176.6	66.4	0.908027

BINNING

Биннинг - это процесс преобразования непрерывных числовых переменных в дискретные категориальные "бины" для проведения группового анализа.

Пример:

В нашем наборе данных "лошадиная сила" - это вещественная переменная в диапазоне от 48 до 288, и она имеет 59 уникальных значений. Что если нас интересует только разница в цене между автомобилями с большой мощностью, средней мощностью и малой мощностью (3 типа)? Можем ли мы разделить их на три "корзины", чтобы упростить анализ?

Мы воспользуемся методом pandas "cut", чтобы разделить столбец "лошадиные силы" на 3 бина.

ПРИВЕДЕМ ДАННЫЕ В НУЖДНЫЙ ТИП

```
df["horsepower"] = df["horsepower"].astype(int, copy=True)
```

```
df['horsepower'].head(20)
```

```
:  0    111
 1    111
 2    154
 3    102
 4    115
Name: horsepower, dtype: int32
```

```
%matplotlib inline
```

```
import matplotlib as plt
```

```
from matplotlib import pyplot
```

```
plt.pyplot.hist(df["horsepower"])
```

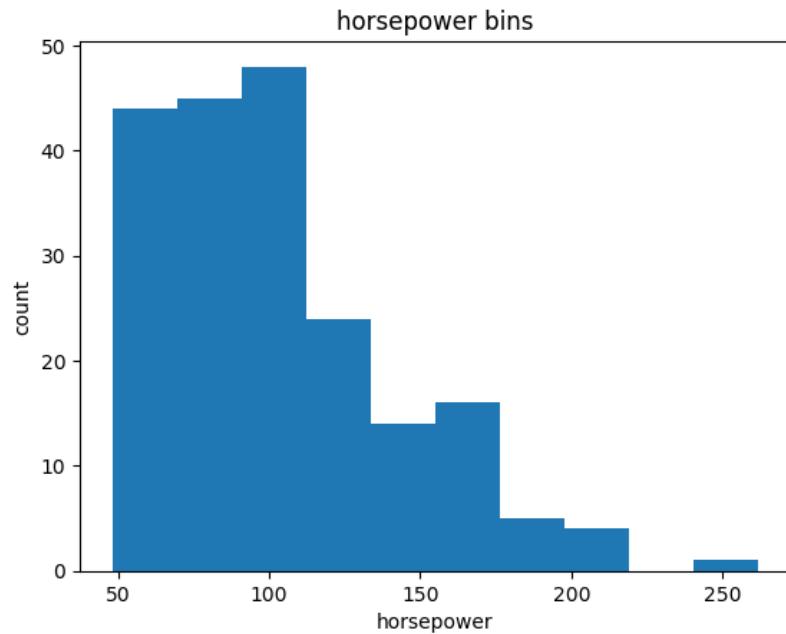
```
# set x/y labels and plot title
```

```
plt.pyplot.xlabel("horsepower")
```

```
plt.pyplot.ylabel("count")
```

```
plt.pyplot.title("horsepower bins")
```

```
Text(0.5, 1.0, 'horsepower bins')
```



Мы хотим получить 3 бина одинакового размера, поэтому используем функцию `linspace(start_value, end_value, numbers_generated)` от numpy.

Поскольку мы хотим включить минимальное значение лошадиной силы, мы хотим установить **start_value = min(df["horsepower"])**.

Поскольку мы хотим включить максимальное значение лошадиных сил, мы хотим установить **end_value = max(df["horsepower"])**.

Поскольку мы строим 3 бина одинаковой длины, должно быть 4 делителя, поэтому **numbers_generated = 4**.

```
bins = np.linspace(min(df["horsepower"]), max(df["horsepower"]), 4)
```

```
bins
```

```
group_names = ['Low', 'Medium', 'High']
```

Мы применяем функцию «cut», чтобы определить, чему принадлежит каждое значение df['лошадиной силы'].

```
df['horsepower-binned'] = pd.cut(df['horsepower'], bins, labels=group_names,  
include_lowest=True )
```

```
df[['horsepower','horsepower-binned']].head(20)
```

	horsepower	horsepower-binned
0	111	Low
1	111	Low
2	154	Medium
3	102	Low
4	115	Low
5	110	Low
6	110	Low
7	110	Low
8	140	Medium
9	101	Low
10	101	Low
11	121	Medium
12	121	Medium
13	121	Medium
14	182	Medium
15	182	Medium
16	182	Medium
17	48	Low
18	70	Low
19	70	Low

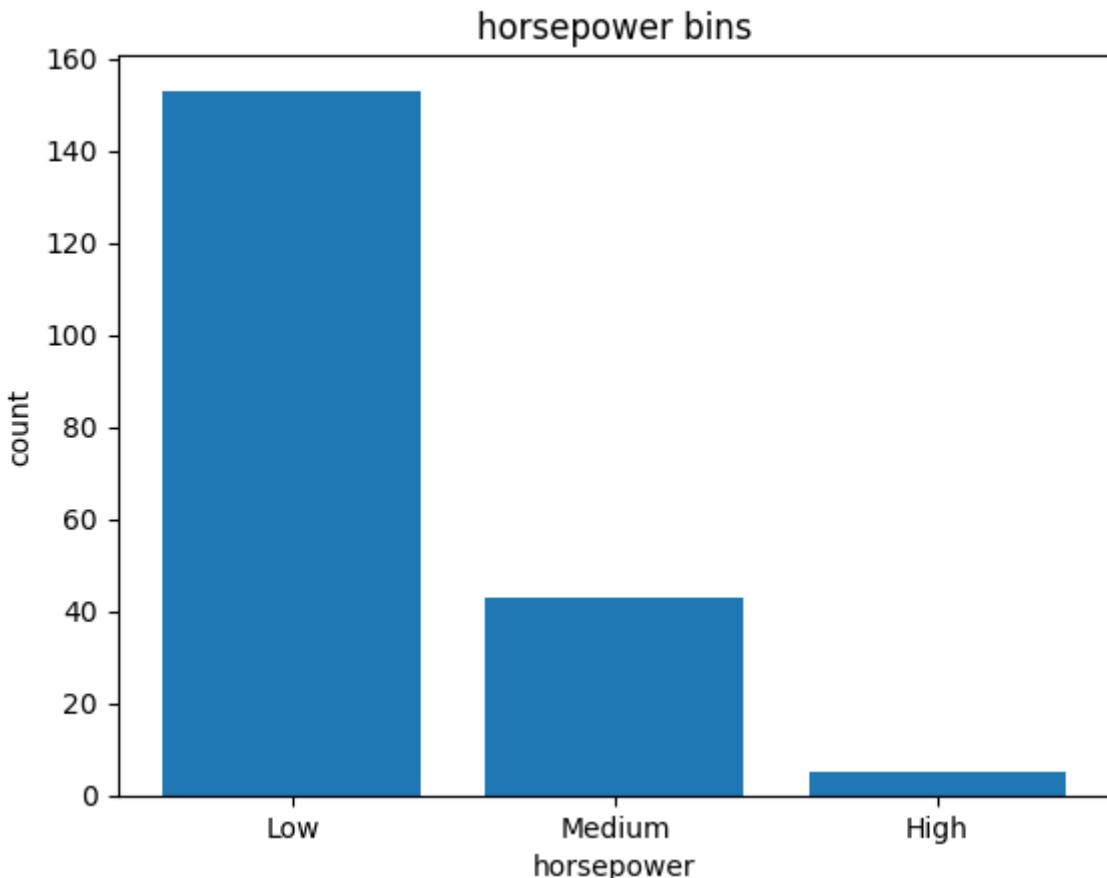
Посмотрим количество автомобилей в каждой корзине:

```
df["horsepower-binned"].value_counts()
```

ПОСТРОИМ ДИСТРИБУТИВНЫЙ ГРАФИК

```
%matplotlib inline  
  
import matplotlib as plt  
  
from matplotlib import pyplot  
  
pyplot.bar(group_names, df["horsepower-binned"].value_counts())  
  
  
# set x/y labels and plot title  
  
plt.pyplot.xlabel("horsepower")  
plt.pyplot.ylabel("count")  
plt.pyplot.title("horsepower bins")
```

```
Text(0.5, 1.0, 'horsepower bins')
```



Внимательно посмотрите на кадр данных выше. Вы обнаружите, что в последнем столбце представлены ячейки для «лошадиных сил» на основе 3 категорий («Низкая», «Средняя» и «Высокая»). Мы успешно сузили интервалы с 59 до 3!

ВИЗУАЛИЗАЦИЯ БИНОВ ОБЫЧНО ГИСТОГРАММА ИСПОЛЬЗУЕТСЯ ДЛЯ ВИЗУАЛИЗАЦИИ РАСПРЕДЕЛЕНИЯ БИНОВ, КОТОРЫЕ МЫ СОЗДАЛИ ВЫШЕ.

```
%matplotlib inline
```

```
import matplotlib as plt
```

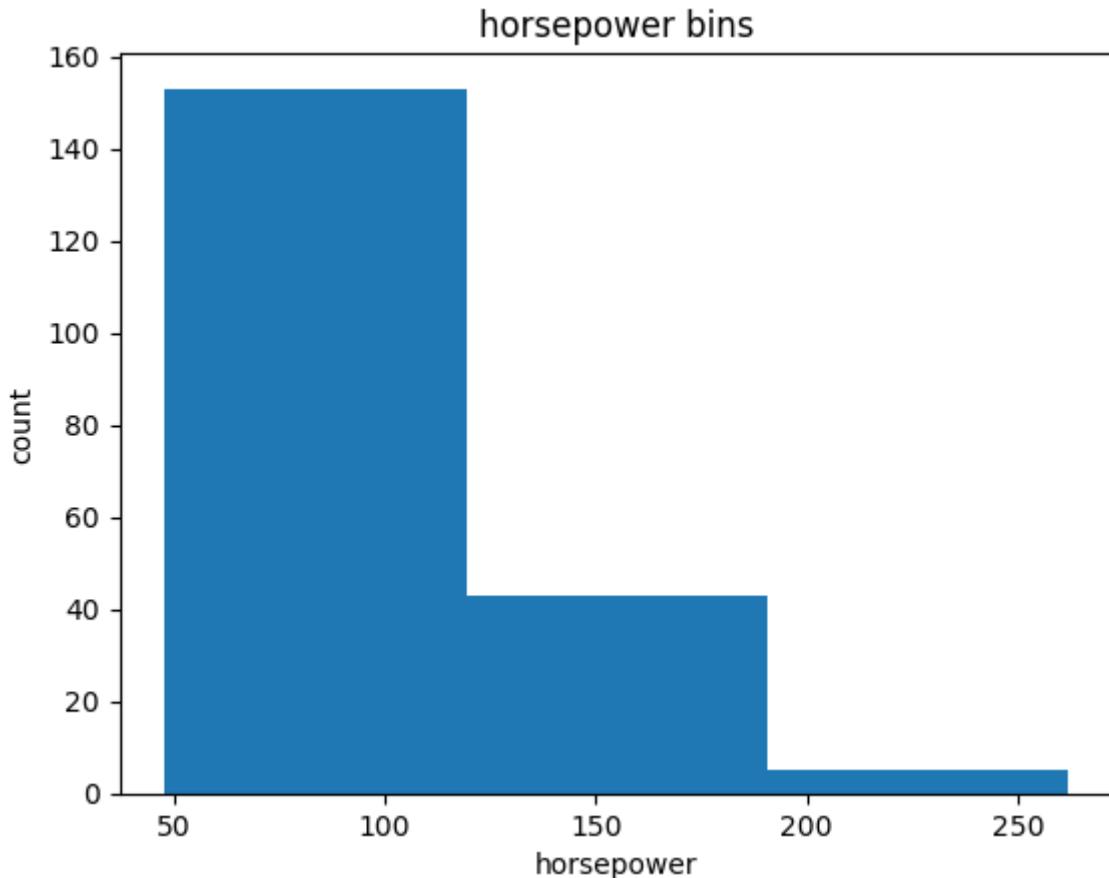
```
from matplotlib import pyplot
```

```
# draw histogram of attribute "horsepower" with bins = 3
```

```
plt.pyplot.hist(df["horsepower"], bins = 3)
```

```
# set x/y labels and plot title
```

```
plt.pyplot.xlabel("horsepower")
plt.pyplot.ylabel("count")
plt.pyplot.title("horsepower bins")
```



ИНДИКАТОРНАЯ ПЕРЕМЕННАЯ (ИЛИ ФИКТИВНАЯ ПЕРЕМЕННАЯ)

Индикаторная переменная (или фиктивная переменная) - это числовая переменная, используемая для обозначения категорий. Их называют "фиктивными", поскольку сами числа не имеют собственного значения.

Мы используем индикаторные переменные, чтобы в последующих модулях мы могли использовать **категориальные переменные для регрессионного анализа**.

Пример

Мы видим, что столбец "тип топлива" имеет два уникальных значения: "газ" или "дизель". Регрессия не понимает слов, только числа. Чтобы использовать этот атрибут в регрессионном анализе, мы преобразуем "тип топлива" в индикаторные переменные.

Мы будем использовать метод pandas '**get_dummies**' для присвоения числовых значений различным категориям типа топлива.

```
df.columns
```

```
dummy_variable_1 = pd.get_dummies(df["fuel-type"])
```

```
dummy_variable_1.head()
```

```
:      diesel   gas
:      ----- 
0       0       1
1       0       1
2       0       1
3       0       1
4       0       1
```

```
dummy_variable_1.rename(columns={'gas':'fuel-type-gas', 'diesel':'fuel-type-diesel'},  
inplace=True)
```

```
dummy_variable_1.head()
```

```
fuel-type-diesel  fuel-type-gas
0               0          1
1               0          1
2               0          1
3               0          1
4               0          1
```

```
# merge data frame "df" and "dummy_variable_1"
```

```
df = pd.concat([df, dummy_variable_1], axis=1)
```

```
# drop original column "fuel-type" from "df"
```

```
df.drop("fuel-type", axis = 1, inplace=True)
```

```
df.head()
```

city-L/100km	highway-l/100km	horsepower-binned	fuel-type-diesel	fuel-type-gas
11.190476	8.703704	Low	0	1
11.190476	8.703704	Low	0	1
12.368421	9.038462	Medium	0	1
9.791667	7.833333	Low	0	1
13.055556	10.681818	Low	0	1

Вопрос №4: Как и раньше, создайте индикаторную переменную для столбца «стремление»

```
# get indicator variables of aspiration and assign it to data frame "dummy_variable_2"
dummy_variable_2 = pd.get_dummies(df['aspiration'])
```

```
# change column names for clarity
```

```
dummy_variable_2.rename(columns={'std':'aspiration-std', 'turbo': 'aspiration-turbo'},
inplace=True)
```

```
# show first 5 instances of data frame "dummy_variable_1"
```

```
dummy_variable_2.head()
```

	aspiration-std	aspiration-turbo
0	1	0
1	1	0
2	1	0
3	1	0
4	1	0

```
# merge the new dataframe to the original datafram
```

```
df = pd.concat([df, dummy_variable_2], axis=1)
```

```
# drop original column "aspiration" from "df"
```

```
df.drop('aspiration', axis = 1, inplace=True)
```

```
df.to_csv('clean_df.csv')
```

city-L/100km	highway-l/100km	horsepower-binned	fuel-type-diesel	fuel-type-gas	aspiration-std	aspiration-turbo
11.190476	8.703704	Low	0	1	1	0
11.190476	8.703704	Low	0	1	1	0
12.368421	9.038462	Medium	0	1	1	0
9.791667	7.833333	Low	0	1	1	0
13.055556	10.681818	Low	0	1	1	0

```
df.describe()
```

price	city-L/100km	highway-l/100km	fuel-type-diesel	fuel-type-gas	aspiration-std	aspiration-turbo
201.000000	201.000000	201.000000	201.000000	201.000000	201.000000	201.000000
13207.129353	9.944145	8.044957	0.099502	0.900498	0.820896	0.179104
7947.066342	2.534599	1.840739	0.300083	0.300083	0.384397	0.384397
5118.000000	4.795918	4.351852	0.000000	0.000000	0.000000	0.000000
7775.000000	7.833333	6.911765	0.000000	1.000000	1.000000	0.000000
10295.000000	9.791667	7.833333	0.000000	1.000000	1.000000	0.000000
16500.000000	12.368421	9.400000	0.000000	1.000000	1.000000	0.000000
45400.000000	18.076923	14.687500	1.000000	1.000000	1.000000	1.000000

ВВЕДЕНИЕ МОДУЛЯ 3

В модуле этой недели вы узнаете, что подразумевается под исследовательским анализом данных, и научитесь выполнять вычисления над данными для расчета основной описательной статистической информации, такой как среднее, медиана, мода и квартильные значения, и использовать эту информацию для лучшего понимания распределения данных. Вы узнаете, как объединять данные в группы для лучшей визуализации данных, как использовать метод корреляции Пирсона для сравнения двух непрерывных числовых переменных, как использовать тест Хи-квадрат для выявления связи между двумя категориальными переменными и как их интерпретировать.

Цели обучения

Описательная статистика

Основы группировки

ANOVA

Корреляция

В этом модуле мы рассмотрим основы разведочного анализа данных с помощью python.

Исследовательский анализ данных или сокращенно EDA - это подход к анализу данных с целью обобщить основные характеристики данных, чтобы лучше понять набор данных, выявить взаимосвязи между различными переменными и извлечь важные переменные для решения проблемы которую мы пытаемся решить.

Основной вопрос, на который мы пытаемся ответить в этом модуле, следующий какие характеристики оказывают наибольшее влияние на цену автомобиля? Мы рассмотрим несколько различных полезных методов анализа данных, чтобы ответить на этот вопрос.

вопрос:

- В этом модуле вы узнаете об описательной статистике, которая описывает основные характеристики набора данных и позволяет получить краткое резюме о выборке и показателях данных.
- Основные группировки данных с помощью группировки по и как это может помочь преобразовать наш набор данных, корреляции между различными переменными, и, наконец, расширенная корреляция. **Здесь мы познакомим вас с различные статистические методы корреляции, а именно корреляция Пирсона и корреляционные тепловые карты**

В этом видео мы поговорим об описательной статистике.

Когда вы начинаете анализировать данные, важно сначала изучить их прежде чем тратить время на построение сложных моделей.

Один из простых способов сделать это - рассчитать некоторые описательные статистики для ваших данных. Описательный статистический анализ помогает описать основные характеристики набора данных и получить краткое резюме о выборке и мерах данных.

Давайте покажем вам несколько различных полезных методов.

Одним из способов, с помощью которого мы можем это сделать, является использование функции `describe` в pandas.

Descriptive Statistics- `Describe()`

- Summarize statistics using pandas `describe()` method

```
df.describe()
```

	Unnamed: 0	symboling	normalized- losses	wheel- base	length	width	height	curb-weight	engine- size	bore	stroke
count	201.000000	201.000000	164.000000	201.000000	201.000000	201.000000	201.000000	201.000000	201.000000	201.000000	201.000000
mean	100.000000	0.840796	122.000000	98.797015	174.200995	65.889055	53.766667	2555.666667	126.875622	3.319154	3.256766
std	58.167861	1.254802	35.442168	6.066366	12.322175	2.101471	2.447822	517.296727	41.546834	0.280130	0.316049
min	0.000000	-2.000000	65.000000	86.600000	141.100000	60.300000	47.800000	1488.000000	61.000000	2.540000	2.070000
25%	50.000000	0.000000	NaN	94.500000	166.800000	64.100000	52.000000	2169.000000	98.000000	3.150000	3.110000
50%	100.000000	1.000000	NaN	97.000000	173.200000	65.500000	54.100000	2414.000000	120.000000	3.310000	3.290000
75%	150.000000	2.000000	NaN	102.400000	183.500000	66.600000	55.500000	2926.000000	141.000000	3.580000	3.410000
max	200.000000	3.000000	256.000000	120.900000	208.100000	72.000000	59.800000	4066.000000	326.000000	3.940000	4.170000

Используя функцию `describe` и применяя ее к вашей рамке данных, функция `describe` автоматически вычисляет базовую статистику для всех числовых переменных.

Она показывает

- среднее значение,
- общее количество точек данных,
- стандартное отклонение,
- квартили
- крайние значения.

Любые значения NAN автоматически пропускаются в этой статистике.

Эта функция даст вам четкое представление о распределении различных переменных.

В вашем наборе данных могут быть также категориальные переменные.

Это переменные, которые могут быть разделены на различные категории или группы и имеют дискретные значения.

Например, в нашем наборе данных система привода является категориальной переменной, которая состоит из категорий: привод на передние колеса, привод на задние колеса и привод на четыре колеса.

Один из способов обобщения категориальных данных, это использование функции `value_counts`.

Descriptive Statistics - Value_Counts()

- summarize the categorical data by using the `value_counts()` method

```
drive_wheels_counts=df[ "drive-wheels" ].value_counts().to_frame()  
drive_wheels_counts.rename(columns={ 'drive-wheels' : 'value_counts' }, inplace=True)  
drive_wheels_counts
```

RM Developer

	value_counts
drive-wheels	
fwd	118
rwd	75
4wd	8

SCOTT'S NETWORK

Мы можем изменить название столбца, чтобы его было легче читать.

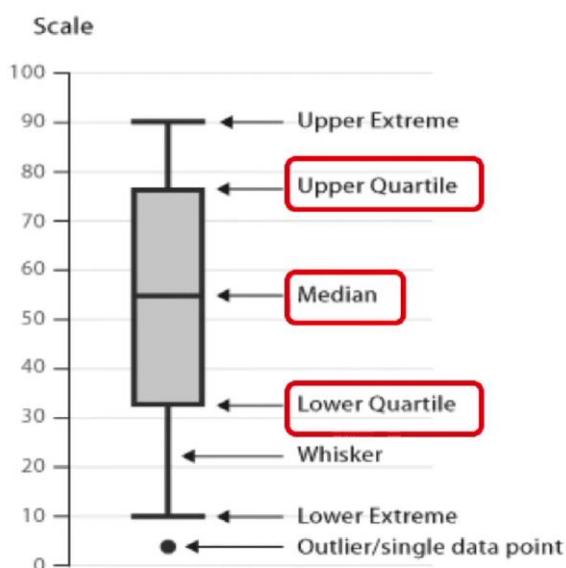
Мы видим, что у нас есть

- 118 автомобилей в категории переднеприводных.
- 75 автомобилей в категории с задним приводом,
- 8 автомобилей в категории с полным приводом.

Коробчатые диаграммы - отличный способ визуализации числовых данных, поскольку вы можете наглядно представить различные распределения данных.

Основными характеристиками, которые отображаются на графике, являются медиана данных, которая показывает, где находится средняя точка данных.

Descriptive Statistics - Box Plots



Верхний квартиль показывает, где находится 75-й процентиль.

Нижний квартиль показывает, где находится 25-й процентиль.

Данные между верхним и нижним квартилем представляют собой интерквартильный размах.

Далее находятся нижний и верхний экстремумы.

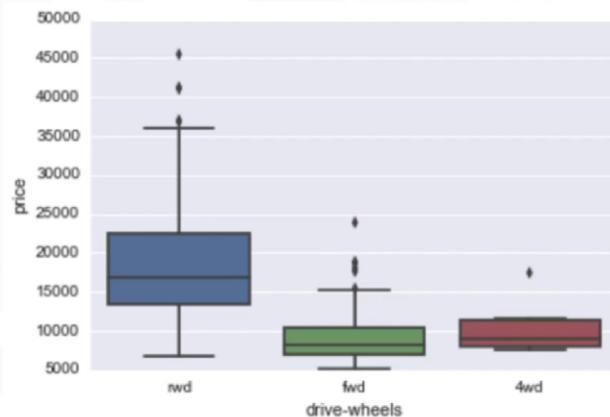
Они рассчитываются как 1,5-кратный интерквартильный размах, выше 75-го процента, и как 1,5-кратный IQR ниже 25-го процента.

Наконец, на коробчатых диаграммах также отображаются отклонения в виде отдельных точек, которые находятся за пределами верхнего и нижнего экстремумов.

С помощью квадратных диаграмм вы можете легко обнаружить выбросы, а также увидеть распределение и перекос данных.

Box Plot - Example

```
sns.boxplot(x= "drive-wheels", y= "price", data=df)
```



С помощью квадратных диаграмм легко сравнивать данные между группами.

В этом примере, используя коробчатую диаграмму, мы можем увидеть распределение различных категорий характеристик ведущих колес по цене.

Мы видим, что распределение цены между задним приводом колес и другими категориями отличается. Но цена для переднеприводных и полный привод практически неотличимы. Часто мы склонны видеть в наших данных **непрерывные переменные**.

Эти точки данных представляют собой числа, находящиеся в некотором диапазоне.

Например, в нашем наборе данных цена и объем двигателя являются непрерывными переменными.

Что если мы хотим понять взаимосвязь между размером двигателя и ценой.

Может ли размер двигателя предсказать цену автомобиля?

Один из хороших способов визуализировать это - использовать диаграмму рассеяния. Каждое наблюдение на диаграмме рассеяния представлено в виде точки. Эта диаграмма показывает взаимосвязь между двумя переменными.

Переменная-предсказатель - это переменная, которую вы используете для прогнозирования результата. В данном случае нашей **предикторной переменной является размер двигателя**. **Целевая переменная** - это переменная, которую вы пытаетесь предсказать. В данном случае целевой переменной является цена.

Descriptive Statistics - Scatter Plot

- Each observation represented as a point.
- Scatter plot Show the relationship between two variables.
 1. Predictor/independent variables on x-axis.
 2. Target/dependent variables on y-axis.

Так как это будет результатом. **В диаграмме рассеяния мы обычно помещаем предикторную переменную на ось x** или горизонтальной оси, а целевую переменную - на оси у или вертикальной оси.

В данном случае мы откладываем размер двигателя по оси x, а цену - по оси y.

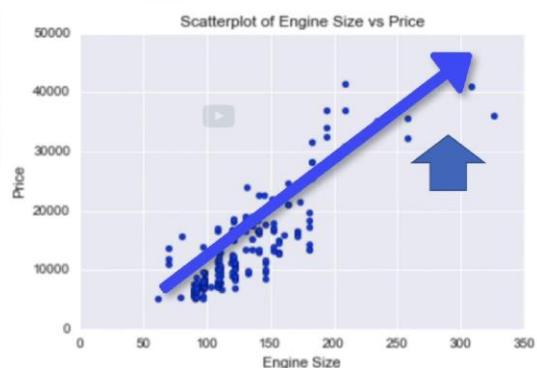
Здесь мы используем функции matplotlib scatter, принимая переменные x и y.

Следует отметить, что всегда важно помечать оси и написать общее название графика, чтобы вы знали, на что смотрите. Как переменная объема двигателя

связана с ценой?

Scatterplot - Example

```
y=df[ "price" ]  
x=df[ "engine-size" ]  
plt.scatter(x,y)  
  
plt.title("Scatterplot of Engine Size vs Price")  
plt.xlabel("Engine Size")  
plt.ylabel("Price")
```



Из диаграммы рассеяния мы видим, что с увеличением объема двигателя, цена автомобиля также увеличивается. Это дает нам первоначальный признак того, что существует положительная линейная зависимость между этими двумя переменными.

ГРУППИРОВКА ДАННЫХ

В этом видео мы рассмотрим основы группировки и как это может помочь преобразовать наш набор данных.

Предположим, вы хотите узнать, есть ли какая-либо связь между различными типами систем привода, передним, задним и полным приводом, и ценой автомобилей?

Если да, то какой тип системы привода добавляет автомобилю наибольшую ценность?

Было бы неплохо, если бы мы могли сгруппировать все данные по различным типам ведущих колес и сравнить результаты этих разных типов ведущих колес друг с другом.

В Pandas это можно сделать с помощью метода **group by**.

Метод group by используется для категориальных переменных, группирует данные в подмножества в соответствии с различными категориями этой переменной.

Вы можете группировать по одной переменной или по нескольким переменным, передавая имена нескольких переменных.

Grouping data

- Use Panda **dataframe. Groupby()** method:
 - Can be applied on categorical variables
 - Group data into categories
 - Single or multiple variables

В качестве примера, допустим, нас интересует поиск средних цен на автомобили и проследить, как они различаются между различными стилями кузова и переменными ведущих колес.

Groupby()- Example

```
df_test = df[['drive-wheels', 'body-style', 'price']]  
df_grp = df_test.groupby(['drive-wheels', 'body-style'], as_index=False).mean()  
df_grp
```

	drive-wheels	body-style	price
0	4wd	convertible	20239.229524
1	4wd	sedan	12647.333333
2	4wd	wagon	9095.750000
3	fwd	convertible	11595.000000
4	fwd	hardtop	8249.000000
5	fwd	hatchback	8396.387755
6	fwd	sedan	9811.800000
7	fwd	wagon	9997.333333
8	rwd	convertible	23949.600000
9	rwd	hardtop	24202.714286
10	rwd	hatchback	14337.777778
11	rwd	sedan	21711.833333
12	rwd	wagon	16994.222222

3M Developer

SKILLS NETWORK

- Для этого сначала выберем три интересующие нас столбца данных, что делается в первой строке кода.
- Затем мы группируем сокращенные данные в соответствии с ведущие колеса и стиль кузова во второй строке.
- Поскольку нас интересует, как различается средняя цена по всем группам, мы можем взять среднее значение для каждой группы и добавить его в самом конце строки.

Теперь данные сгруппированы по подкатегориям и отображается только средняя цена каждой подкатегории.

Мы видим, что согласно нашим данным, заднеприводные кабриолеты и заднеприводные хардтопы имеют самую высокую стоимость, в то время как хэтчбеки с приводом на четыре колеса имеют самую низкую стоимость.

Таблица такой формы не самая удобная для чтения, а также не очень проста для визуализации.

Чтобы сделать ее более понятной, мы можем преобразовать эту таблицу в поворотную таблицу с помощью метода **pivot**.

Pandas method - Pivot()

- One variable displayed along the columns and the other variable displayed along the rows.

```
df_pivot = df_grp.pivot(index= 'drive-wheels', columns='body-style')
```

	price				
body-style	convertible	hardtop	hatchback	sedan	wagon
drive-wheels					
4wd	20239.229524	20239.229524	7603.000000	12647.333333	9095.750000
fwd	11595.000000	8249.000000	8396.387755	9811.800000	9997.333333
rwd	23949.600000	24202.714286	14337.777778	21711.833333	16994.222222

В предыдущей таблице и ведущие колеса, и стиль кузова были слуховыми колонками.

В поворотной таблице одна переменная отображается вдоль столбцов, а другая переменная отображается по строкам.

Всего одной строкой кода и с помощью метода `pivot` Panda, мы можем повернуть переменную стиля кузова так, чтобы она отображалась вдоль столбцов, а ведущие колеса будут отображаться вдоль строк.

Теперь данные о ценах становятся прямоугольной сеткой, что проще для визуализации.

Это похоже на то, что обычно делается в электронных таблицах Excel.

Другой способ представления таблицы `pivot` - использование графика тепловой карты. Термальная карта представляет собой прямоугольную сетку данных и назначает интенсивность цвета в зависимости от значения данных в точках сетки.

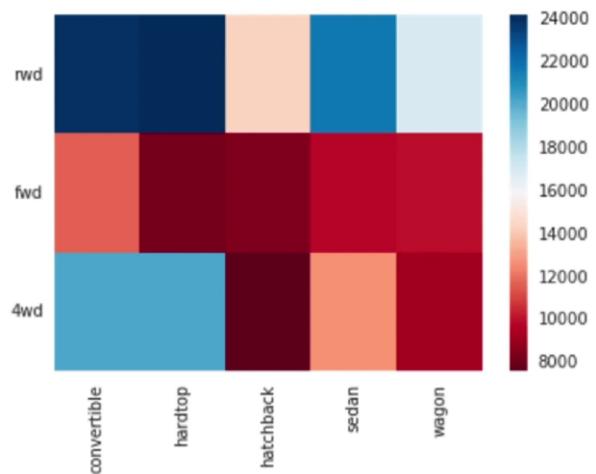
Это отличный способ построить график целевой переменной по нескольким переменным и благодаря этому получить визуальные подсказки о связи между этими переменными и целевой переменной.

В этом примере мы используем метод `p color в pyplot`, чтобы построения тепловой карты и преобразования предыдущей таблицы `pivot` в графическую форму.

Heatmap

- Plot target variable over multiple variables

```
plt.pcolor(df_pivot, cmap='RdBu')
plt.colorbar()
plt.show()
```



Мы задаем красно-синюю (**RdBu**) цветовую схему.

В выходном графике каждый тип кузова пронумерован вдоль по оси x, а каждый тип ведущих колес пронумерован по оси y. Средние цены отображаются на графике разными цветами в зависимости от их значений. В соответствии с цветовой шкалой, мы видим, что в верхней части тепловой карты имеет более высокие цены, чем нижний участок.

КОРРЕЛЯЦИЯ

В этом видео мы поговорим о корреляции между различными переменными.

Корреляция - это статистическая метрика для измерения степени взаимозависимости различных переменных.

Другими словами, когда мы смотрим на две переменные во времени, если одна переменная изменяется, как это влияет на изменение другой переменной?

Например, известно, что курение коррелирует с раком легких поскольку у вас больше шансов заболеть раком легких, если вы курите.

В другом примере существует корреляция между зонтиком и переменными дождя, где больше осадков означает, что больше людей пользуются зонтиками.

Кроме того, если бы не было дождя, люди не носили бы зонтики.

Поэтому мы можем сказать, что зонтики и дождь являются взаимозависимы и по определению коррелируют.

Важно знать, что корреляция не означает причинно-следственную связь.

Фактически, мы можем сказать, что зонт и дождь коррелируют, но у нас не будет достаточно информации, чтобы сказать, коррелируют ли зонт и дождь.

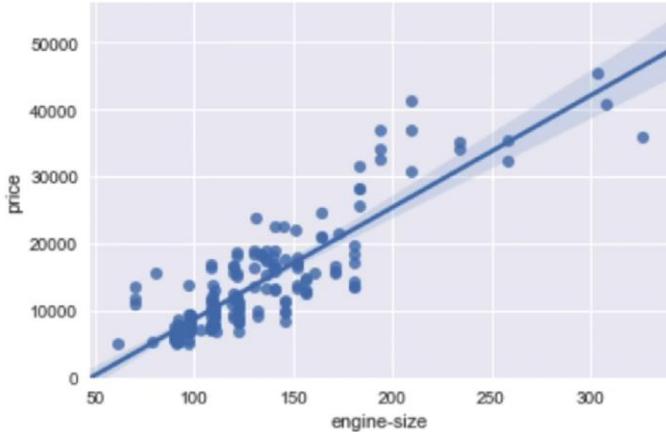
достаточно информации, чтобы сказать, зонтик ли вызвал дождь или дождь вызвал зонт. В науке о данных мы обычно больше имеем дело с корреляцией. Давайте рассмотрим корреляцию между размером двигателя и ценой. **На этот раз мы визуализируем эти две переменные с помощью диаграммы рассеяния и добавленной линейной линии, называемой линией регрессии, которая показывает взаимосвязь между этими двумя переменными.**

Основная цель этого графика - увидеть, влияет ли размер двигателя на цену.

Correlation - Positive Linear Relationship

- Correlation between two features (engine-size and price).

```
sns.regplot(x="engine-size", y="price", data=df)  
plt.ylim(0, )
```



В этом примере видно, что прямая линия, проходящая через точки данных, очень крутая, что свидетельствует о наличии положительной линейной зависимости между двумя переменными.

С увеличением значения размера двигателя, значение цены также увеличивается, и наклон линии положительный. Таким образом, существует положительная корреляция между размером двигателя и ценой.

Мы можем использовать **seaborn.regplot** для построения диаграммы рассеяния.

В качестве другого примера, теперь давайте рассмотрим взаимосвязь между количеством миль на галлон по шоссе, чтобы увидеть его влияние на цену автомобиля.

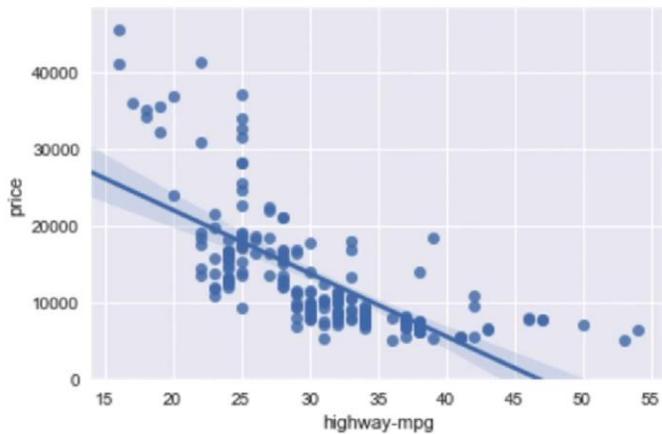
Как мы видим на этом графике, когда значение показателя "миль на галлон" растет, цена автомобиля падает.

Поэтому существует отрицательная линейная зависимость между количеством миль на галлон по шоссе и ценой.

Correlation - Negative Linear Relationship

- Correlation between two features (highway-mpg and price).

```
sns.regplot(x="highway-mpg", y="price", data=df)  
plt.ylim(0,)
```



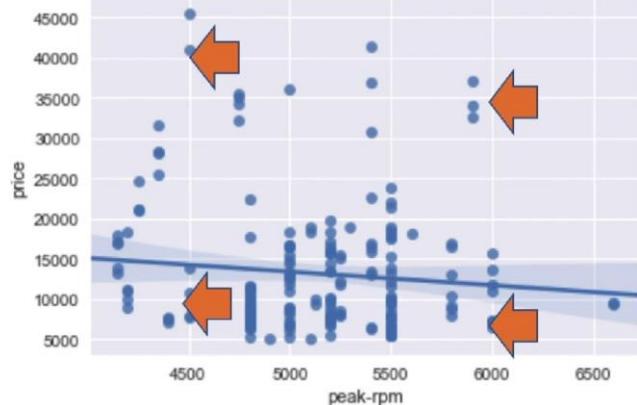
Хотя эта зависимость отрицательная, наклон линии крутой, что означает, что количество миль на галлон по шоссе все еще является хорошим предиктором цены. Считается, что эти две переменные имеют отрицательную корреляцию.

Наконец, у нас есть пример слабой корреляции.

Correlation - Negative Linear Relationship

- Weak correlation between two features (peak-rpm and price).

```
sns.regplot(x="peak-rpm", y="price", data=df)  
plt.ylim(0,)
```



Например, как низкие значения пикового числа оборотов, так и высокие значения пикового числа оборотов имеют низкие и высокие цены.

Поэтому мы не можем использовать RPM для прогнозирования цен.

Correlation – Statistics

Один из способов измерения силы корреляции между непрерывной числовой переменной является использование метода, **называемого корреляцией Пирсона.**

Метод корреляции Пирсона дает вам два значения: **коэффициент корреляции и Р-значение.**

Как же интерпретировать эти значения?

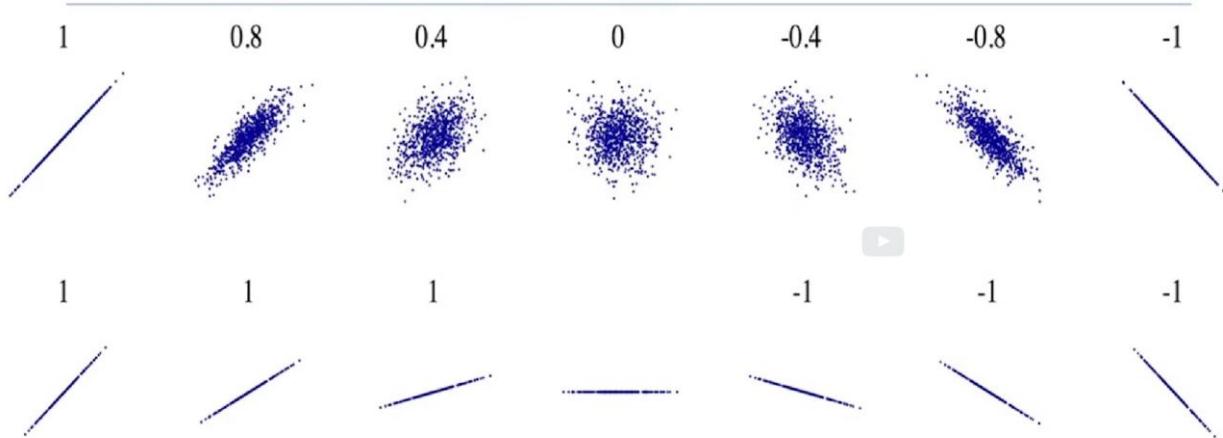
- Для коэффициента корреляции, значение, близкое к 1, **подразумевает большую положительную корреляцию**, в то время как значение, близкое к -1, означает большую отрицательную корреляцию, а значение, близкое к нулю, означает отсутствие корреляции между переменными.
- Далее, **P-значение** покажет нам, насколько мы уверены в корреляции, которую мы вычислили.
- Для Р-значения значение меньше, чем 0.001 дает нам сильную уверенность в том, что мы вычислили коэффициент корреляции.
- Значение от 0.001 до 0.05 дает нам умеренную уверенность. Значение между 0.05 и 0.1 дает нам слабую уверенность.
- А значение Р-значения больше, чем 0.1, не даст нам никакой уверенности в наличии корреляции вообще.
- Мы можем сказать, что существует сильная корреляция, когда коэффициент корреляции близок к 1 или отрицателен 1, а Р-значение меньше, чем 0.001.

Pearson Correlation

- Measure the strength of the correlation between two features.
 - Correlation coefficient
 - P-value
- Correlation coefficient
 - Close to +1: Large Positive relationship
 - Close to -1: Large Negative relationship
 - Close to 0: No relationship
- P-value
 - P-value < 0.001 **Strong** certainty in the result
 - P-value < 0.05 **Moderate** certainty in the result
 - P-value < 0.1 **Weak** certainty in the result
 - P-value > 0.1 **No** certainty in the result
- Strong Correlation:
 - Correlation coefficient close to 1 or -1
 - P value less than 0.001

На следующем графике показаны данные с различными значениями корреляции.

Pearson Correlation



В этом примере мы хотим рассмотреть корреляцию между переменной "лошадиная сила" и ценой автомобиля.

Видите, как легко можно рассчитать корреляцию Пирсона с помощью пакета статистики SI/PI?

Pearson Correlation- Example

```
pearson_coef, p_value = stats.pearsonr(df['horsepower'], df['price'])
```

- Pearson correlation: 0.81
- P-value : 9.35 e-48

Мы видим, что коэффициент корреляции равен приблизительно 0.8, а это близко к 1.

Таким образом, существует сильная положительная корреляция. Мы также видим, что Р-значение очень мало, намного меньше, чем 0.001.

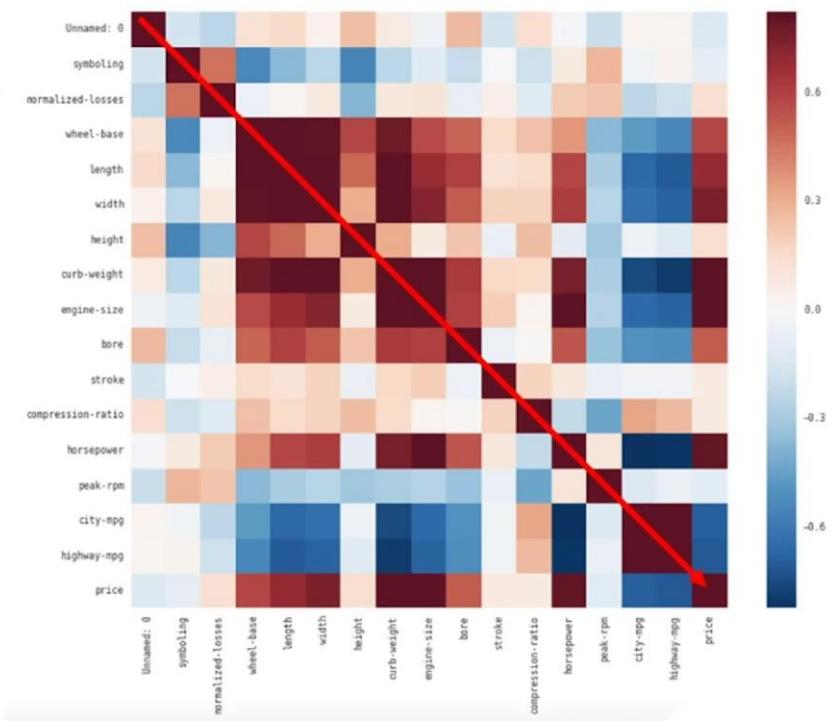
Таким образом, мы можем сделать вывод, что мы уверены в сильной положительной корреляции.

Принимая во внимание все переменные, теперь мы можем создать тепловую карту, которая показывает корреляцию между каждой из переменных друг с другом.

Цветовая схема указывает на коэффициент корреляции Пирсона, указывающий на силу корреляции между двумя переменными.

Мы видим диагональную линию **темно-красного цвета**, указывающую на то, что все значения на этой диагонали сильно коррелируют.

Correlation-Heatmap



Это имеет смысл, потому что, если присмотреться, значения на диагонали - это корреляция всех переменных между собой, которая всегда равна 1.

Эта тепловая карта корреляции дает нам хорошее представление о том, как различные переменные связаны друг с другом и, самое главное, как эти переменные связаны с ценой.

ТЕСТЫ ХИ-КВАДРАТ НА НЕЗАВИСИМОСТЬ

Недавно мы узнали о корреляционном тесте Пирсона для двух непрерывных переменных. В некоторых случаях наши данные могут содержать категориальные переменные - для проверки взаимосвязи между двумя категориальными переменными. **Хи-квадрат проверяет, как одна категориальная переменная влияет на другую категориальную переменную, используя распределение частот в каждой группе.**

Мы будем использовать набор данных "Автомобили". **Предположим**, что мы хотим проверить взаимосвязь между типом топлива и мощностью двигателя. Это две категориальные переменные, и у них нет непрерывных точек, к которым они могли бы привязаться. Либо это стандартный автомобиль на дизельном топливе, либо стандартный автомобиль на газовом топливе, либо турбо автомобиль на дизельном топливе, либо турбо автомобиль на газовом топливе. Мы хотим узнать, связан ли тип топлива с мощностью двигателя автомобиля. Перед этим давайте пройдемся по некоторым важным моментам:

Хи-квадрат проверяет нулевую гипотезу о том, что переменные независимы.

Тест сравнивает наблюдаемые данные со значениями, которые ожидает модель, если бы данные были распределены по разным категориям случайно. **Если наблюдаемые данные не вписываются в модель ожидаемых значений, вероятность того, что переменные являются зависимыми, возрастает, тем самым доказывая, что нулевая гипотеза неверна.**

Хи-квадрат не говорит вам о типе связи, которая существует между обеими переменными, только о том, что связь существует.

Теперь давайте проверим, существует ли связь между типом топлива и мощностью двигателя.

Сначала мы найдем наблюдаемые значения автомобилей в каждой категории. Это можно сделать с помощью **crosstab** в библиотеке pandas.

	Standard	Turbo	Total
diesel	7	13	20
gas	161	24	185
Total	168	37	205

Затем мы рассчитаем ожидаемые значения, т.е. значения, которые мы ожидаем, если бы каждый автомобиль попадал в категорию случайным образом. Для этого мы используем следующую формулу:

$$\frac{\text{RowTotal} * \text{ColumnTotal}}{\text{GrandTotal}}$$

Используя приведенную выше формулу, ожидаемые значения для каждой группы будут следующими:

Стандартный автомобиль с дизельным топливом = $(20 * 168) / 205$

Стандартный автомобиль на газовом топливе = $(185 * 168) / 205$

Турбо автомобиль с дизельным топливом = $(20 * 37) / 205$ и

Турбо автомобиль на газовом топливе = $(185 * 37) / 205$

Мы также можем посмотреть на ожидаемые значения в виде перекрестной таблицы:

	Standard	Turbo	Total
diesel	16.39	3.61	20
gas	151.61	33.39	185
Total	168	37	205

Мы видим, что ожидает модель, а также видим, что промежуточные и итоговые значения равны наблюдаемым значениям. Теперь мы проведем тест Хи-квадрат. Формула выглядит следующим образом:

$$\chi^2 = \sum_{k=1}^n \frac{(O_i - E_i)^2}{E_i}$$

Это даст значение хи-квадрат χ^2 , которое в данном случае равно 29,6. Мы воспользуемся таблицей хи-квадрат и найдем соответствующее р-значение, для чего найдем степень независимости (строка-1)*(столбец-1) = 1, а затем соответствующее р-значение 29,6. Используя таблицу хи-квадрат, мы увидим, что р-значение очень близко к 0. Это означает, что существует связь между типом топлива и мощностью двигателя.

#Это можно сделать в python с помощью пакета scipy.stats. Сначала мы создадим перекрестную таблицу

```
cont_table = pd.crosstab(df['fuel-type'], df['aspiration'])
```

затем используем хи2_контингент из библиотеки scipy.stats

```
scipy.stats.chi2_contingency(cont_table, correction = True)
```

Это выведет статистику хи-квадрат, р-значение, степень свободы и ожидаемые значения.

ИТОГО

Описывать эксплораторный анализ данных: Обобщать основные характеристики данных и извлекать ценные сведения.

Вычислять основные описательные статистики: Вычислять среднее значение, медиану и моду с помощью python и использовать их в качестве основы для понимания распределения данных.

Создание групп данных: как и зачем объединять непрерывные данные в группы и как их визуализировать.

Определите корреляцию как линейную связь между двумя числовыми переменными: Используйте корреляцию Пирсона в качестве меры корреляции между двумя непрерывными переменными.

Определите ассоциацию между двумя категориальными переменными:

Понять, как найти связь между двумя переменными с помощью теста Хи-квадрат для ассоциации и как их интерпретировать.

ЛАБОРАТОРНАЯ РАБОТА

#you are running the lab in your browser, so we will install the libraries using ``piplite``

```
import piplite  
  
await piplite.install(['pandas'])  
  
await piplite.install(['matplotlib'])  
  
await piplite.install(['scipy'])  
  
await piplite.install(['seaborn'])
```

#If you run the lab locally using Anaconda, you can load the correct library and versions by uncommenting the following:

```
#install specific version of libraries used in lab  
  
#! mamba install pandas==1.3.3  
  
#! mamba install numpy=1.21.2  
  
#! mamba install scipy=1.7.1-y  
  
#! mamba install seaborn=0.9.0-y
```

```
import pandas as pd  
  
import numpy as np
```

#This function will download the dataset into your browser

```
from pyodide.http import pyfetch

async def download(url, filename):

    response = await pyfetch(url)

    if response.status == 200:

        with open(filename, "wb") as f:

            f.write(await response.bytes())
```

```
path='https://cf-courses-data.s3.us.cloud-object-
storage.appdomain.cloud/IBMDriverSkillsNetwork-DA0101EN-
SkillsNetwork/labs/Data%20files/automobileEDA.csv'
```

```
await download(path, "auto.csv")

filename="auto.csv"
```

```
df = pd.read_csv(filename)
```

```
df.head()
```

	symboling	normalized-losses	make	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	length	...	compression-ratio	horsepower	peak-rpm	city-mpg
0	3	122	alfa-romero	std	two	convertible	rwd	front	88.6	0.811148	...	9.0	111.0	5000.0	21
1	3	122	alfa-romero	std	two	convertible	rwd	front	88.6	0.811148	...	9.0	111.0	5000.0	21
2	1	122	alfa-romero	std	two	hatchback	rwd	front	94.5	0.822681	...	9.0	154.0	5000.0	19
3	2	164	audi	std	four	sedan	fwd	front	99.8	0.848630	...	10.0	102.0	5500.0	24
4	2	164	audi	std	four	sedan	4wd	front	99.4	0.848630	...	8.0	115.0	5500.0	18

```
import matplotlib.pyplot as plt
```

```
import seaborn as sns
```

```
%matplotlib inline
```

```
# list the data types for each column
```

```
print(df.dtypes)
```

```

symboling           int64
normalized-losses   int64
make                object
aspiration          object
num-of-doors        object
body-style          object
drive-wheels        object
engine-location     object
wheel-base          float64
length              float64
width               float64
height              float64
curb-weight         int64
engine-type         object
num-of-cylinders    object
engine-size          int64
fuel-system          object
bore                 float64
stroke              float64
compression-ratio   float64
horsepower          float64
peak-rpm             float64
city-mpg             int64
highway-mpg          int64
price                float64
city-L/100km         float64
horsepower-binned    object
diesel               int64
gas                  int64
dtype: object

```

Узнать тип данных конкретного столбца

`df['peak-rpm'].dtypes`

Например, мы можем вычислить корреляцию между переменными типа «int64» или «float64», используя метод «corr»:

	symboling	normalized-losses	wheel-base	length	width	height	curb-weight	engine-size	bore	stroke	compression-ratio	horsepower	peak-rpm	city-mpg	highway-mpg	price	city-L/100km	diesel	gas
symboling	1.000000	0.466264	-0.535987	-0.365404	-0.242423	-0.550160	-0.233118	-0.110581	-0.140019	-0.008245	-0.182196	0.075819	0.297940	-0.035527	0.036233	-0.082391	0.061711	-0.196735	
normalized-losses	0.466264	1.000000	-0.056661	0.019424	0.086802	-0.073737	0.099404	0.112360	-0.029862	0.055563	-0.114713	0.217299	0.239543	-0.225016	-0.181877	0.133999	0.238567	-0.101546	
wheel-base	-0.535987	-0.056661	1.000000	0.876024	0.814507	0.590742	0.782097	0.572027	0.493244	0.158502	0.250313	0.371147	-0.360305	-0.470606	-0.543304	0.584642	0.476153	0.307237	
length	-0.365404	0.019424	0.876024	1.000000	0.857170	0.492063	0.880665	0.685025	0.608971	0.124139	0.159733	0.579821	-0.285970	-0.665192	-0.698142	0.690628	0.657373	0.211187	
width	-0.242423	0.086802	0.814507	0.857170	1.000000	0.306002	0.866201	0.729436	0.544885	0.188829	0.189867	0.615077	-0.245800	-0.633531	-0.680635	0.751265	0.673363	0.244356	
height	-0.550160	-0.373737	0.590742	0.492063	0.306002	1.000000	0.307581	0.074694	0.180449	-0.062704	0.259737	-0.087027	-0.309974	-0.049800	-0.104812	0.135486	0.003811	0.281578	
curb-weight	-0.233118	0.099404	0.782097	0.880665	0.866201	0.307581	1.000000	0.849972	0.644065	0.167562	0.156433	0.757976	-0.279561	-0.749543	-0.794889	0.834415	0.785353	0.221046	
engine-size	-0.110581	0.112360	0.572027	0.685025	0.729436	0.074694	0.849072	1.000000	0.572609	0.209523	0.028889	0.822676	-0.256733	-0.650546	-0.679571	0.872335	0.745059	0.070779	
bore	-0.140019	-0.029862	0.493244	0.609971	0.544885	0.180449	0.644060	0.572609	1.000000	-0.055390	0.001263	0.566936	-0.267592	-0.582027	-0.591309	0.543155	0.554610	0.054458	
stroke	-0.008245	0.055563	0.158502	0.124139	0.188829	-0.062704	0.167562	0.209523	-0.055390	1.000000	0.187923	0.098462	-0.065713	-0.034696	-0.035201	0.082310	0.037300	0.241303	
compression-ratio	-0.182196	-0.114713	0.250313	0.159733	0.189867	0.259737	0.156433	0.028889	0.001263	0.187923	1.000000	-0.214514	-0.435780	0.331425	0.268465	0.071107	-0.299372	0.985231	
horsepower	0.075819	0.217299	0.371147	0.579821	0.615077	-0.087027	0.757976	0.822676	0.566936	0.098462	-0.214514	1.000000	0.107885	-0.822214	-0.804575	0.809575	0.889488	-0.169053	0.169053
peak-rpm	0.279740	0.239543	-0.360305	-0.285970	-0.245800	-0.309974	-0.279361	-0.256733	-0.267392	-0.065713	-0.425780	0.107885	1.000000	-0.115413	-0.058598	-0.101616	0.115830	-0.475812	
city-mpg	-0.035527	-0.225016	-0.470606	-0.665194	-0.633331	-0.049800	-0.749543	-0.650546	-0.582027	-0.034696	0.331425	-0.822214	-0.115413	1.000000	0.972044	-0.686571	-0.949713	0.265676	-0.265676
highway-mpg	0.036233	-0.181877	-0.543304	-0.698142	-0.680635	-0.104012	-0.794889	-0.679571	-0.591309	-0.035201	0.268465	-0.804575	-0.058598	0.972044	1.000000	-0.704692	-0.930028	0.198690	-0.198690
price	-0.082391	0.133999	0.584642	0.699620	0.751265	0.135486	0.834415	0.872335	0.543155	0.082310	0.071107	0.809575	-0.101616	-0.686571	-0.704692	1.000000	0.789898	0.110326	
city-L/100km	0.066171	0.238567	0.476153	0.657373	0.673363	0.003811	0.785353	0.745059	0.554610	0.037300	-0.299372	0.889488	0.115830	-0.949713	-0.930028	0.789898	1.000000	-0.241282	
diesel	-0.196735	-0.101546	0.307237	0.211187	0.244356	0.281578	0.221046	0.070779	0.054458	-0.241303	0.985231	-0.169053	-0.475812	0.265676	0.198690	0.110326	-0.241282	1.000000	
gas	0.196735	0.101546	-0.307237	-0.211187	-0.244356	-0.281578	-0.221046	-0.070779	-0.054458	-0.241303	-0.985231	0.169053	0.475812	-0.265676	-0.198690	-0.110326	0.241282	-1.000000	

Диагональные элементы всегда одни; мы более подробно изучим корреляцию, корреляцию Пирсона, в конце блокнота

Вопрос 2: Найдите соотношение между следующими столбцами: диаметр цилиндра, ход поршня, степень сжатия и мощность. Подсказка: если вы хотите выбрать эти столбцы, используйте следующий синтаксис:
`df[['bore','stroke','compression-ratio','horsepower']].corr`

	bore	stroke	compression-ratio	horsepower
bore	1.000000	-0.055390	0.001263	0.566936
stroke	-0.055390	1.000000	0.187923	0.098462
compression-ratio	0.001263	0.187923	1.000000	-0.214514
horsepower	0.566936	0.098462	-0.214514	1.000000

Непрерывные числовые переменные:

Непрерывные числовые переменные - это переменные, которые могут содержать любое значение в некотором диапазоне. Они могут иметь тип "int64" или "float64".

Отличным способом визуализации этих переменных является использование диаграмм рассеяния с подогнанными линиями.

Для того чтобы начать понимать (линейную) связь между отдельной переменной и ценой, мы можем использовать "regplot", который строит график рассеяния плюс подогнанную линию регрессии для данных.

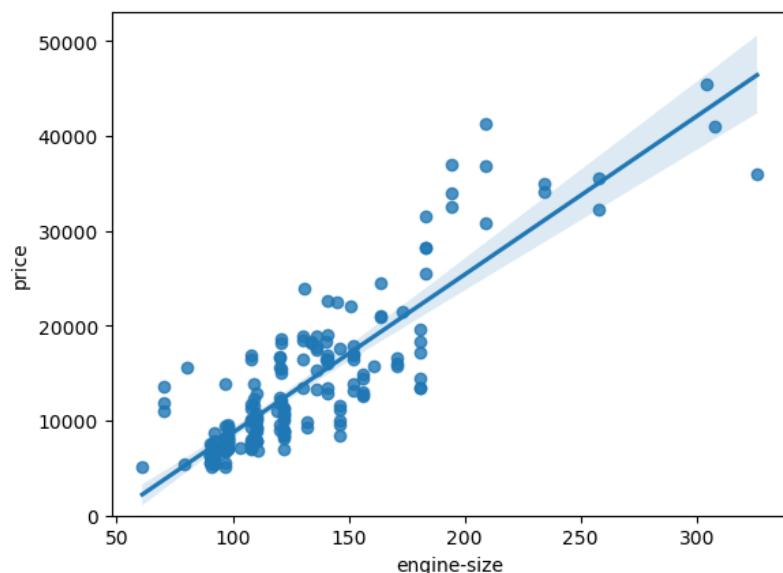
Давайте посмотрим несколько примеров различных линейных отношений:

Engine size as potential predictor variable of price

```
sns.regplot(x="engine-size", y="price", data=df)
```

```
plt.ylim(0,)
```

```
(0.0, 53058.86937911297)
```



По мере увеличения объема двигателя растет и цена: это указывает на прямую положительную корреляцию между этими двумя переменными. Объем двигателя кажется довольно хорошим предсказателем цены, поскольку линия регрессии представляет собой почти идеальную диагональную линию.

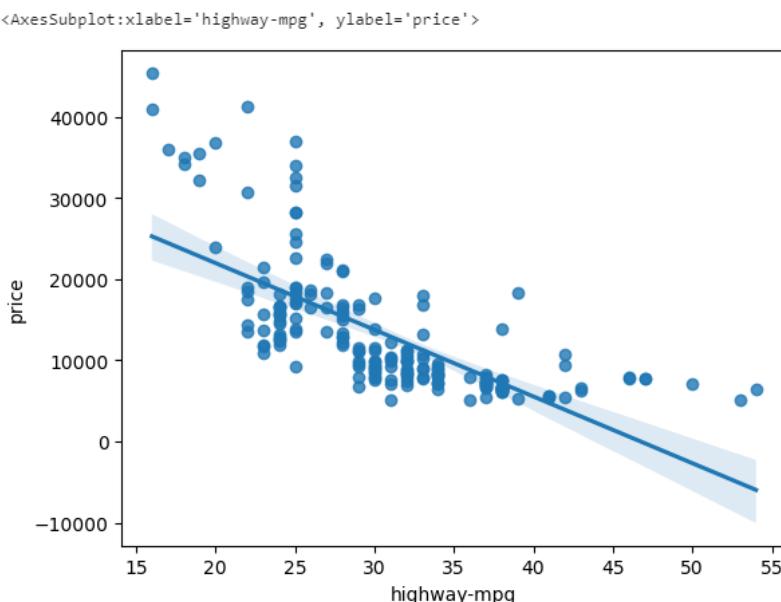
Мы можем изучить корреляцию между «объемом двигателя» и «ценой» и увидеть, что она составляет примерно 0,87.

```
df[["engine-size", "price"]].corr()
```

	engine-size	price
engine-size	1.000000	0.872335
price	0.872335	1.000000

Миля на галлон по шоссе — это потенциальный предиктор цены. Давайте найдем диаграмму рассеяния «шоссе-миль на галлон» и «цена».

```
sns.regplot(x="highway-mpg", y="price", data=df)
```



Когда шоссе-миль на галлон растет, цена падает: это указывает на обратную/отрицательную связь между этими двумя переменными. Расход на галлон по шоссе потенциально может быть предиктором цены

Мы можем изучить корреляцию между «миль на галлон» и «ценой» и увидеть, что она составляет приблизительно -0,704.

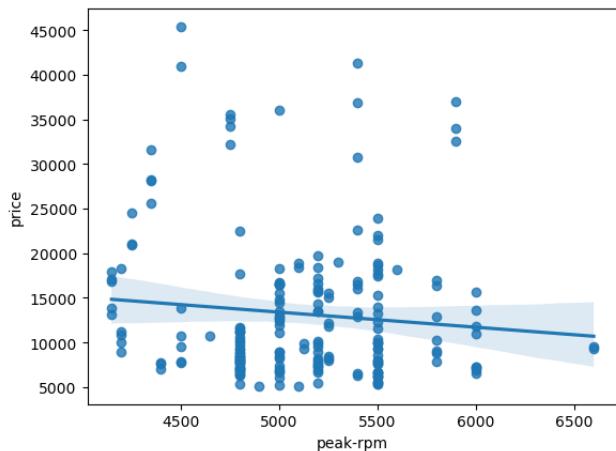
```
df[['highway-mpg', 'price']].corr()
```

	highway-mpg	price
highway-mpg	1.000000	-0.704692
price	-0.704692	1.000000

СЛАБАЯ ЛИНЕЙНАЯ СВЯЗЬ

Давайте посмотрим, является ли «пик-об/мин» переменной-предиктором «цены».

```
sns.regplot(x="peak-rpm", y="price", data=df)
```



Максимальные обороты вообще не являются хорошим предсказателем цены, поскольку линия регрессии близка к горизонтальной. Кроме того, точки данных сильно разбросаны и находятся далеко от подогнанной линии, что свидетельствует о большой изменчивости. Следовательно, это ненадежная переменная

Мы можем изучить корреляцию между «пиковыми оборотами» и «ценой» и увидеть, что она составляет примерно -0,101616

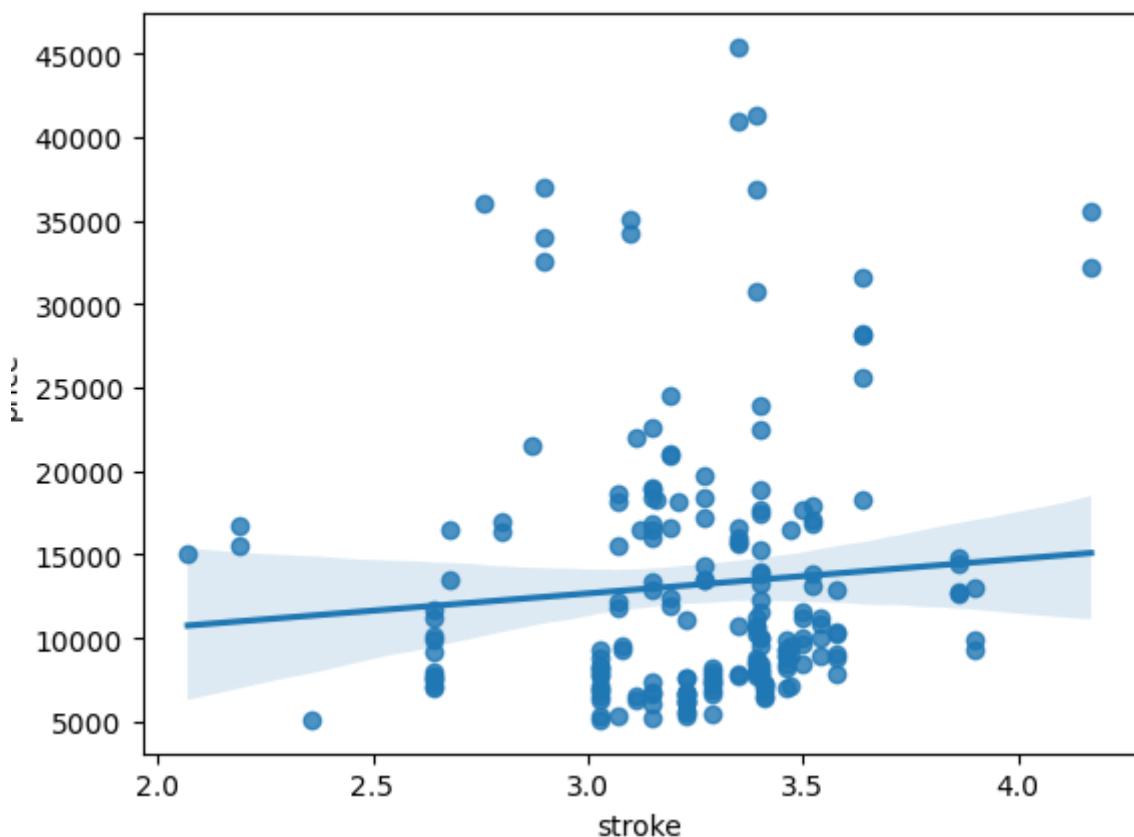
```
df[['peak-rpm','price']].corr()
```

Найдите корреляцию между x="ход" и y="цена". Подсказка: если вы хотите выбрать эти столбцы, используйте следующий синтаксис: df[["stroke","price"]].

```
sns.regplot(x="stroke", y="price", data = df)
```

```
df[["stroke","price"]].corr()
```

	stroke	price
stroke	1.000000	0.08231
price	0.08231	1.000000

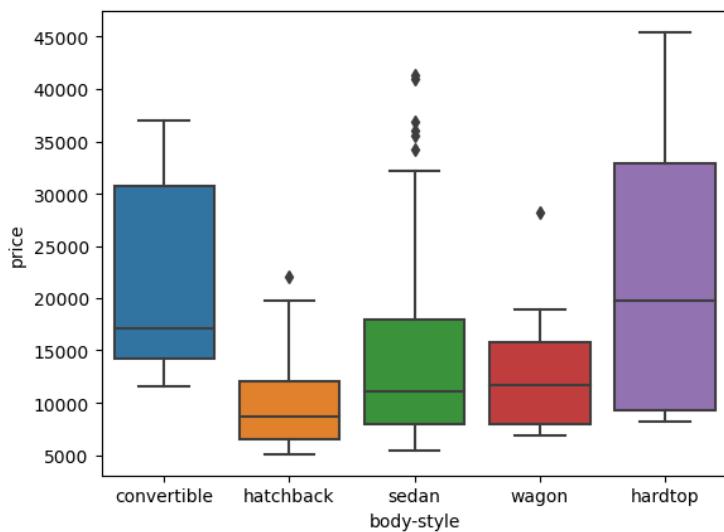


CATEGORICAL VARIABLES

Это переменные, которые описывают "характеристику" единицы данных и выбираются из небольшой группы категорий. Категориальные переменные могут иметь тип "объект" или "int64". Хорошим способом визуализации категориальных переменных является использование бокфлотов

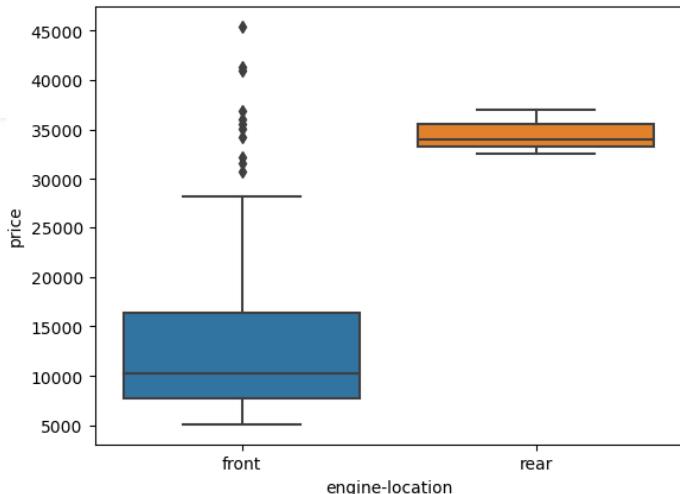
Давайте посмотрим на взаимосвязь между «типов кузова» и «ценой».

```
sns.boxplot(x="body-style", y="price", data=df)
```



Мы видим, что распределения цен между различными категориями телосложения значительно перекрываются, поэтому телосложение не может быть хорошим предиктором цены. Рассмотрим двигатель "двигатель-местоположение" и "цена":

```
sns.boxplot(x="engine-location", y="price", data=df)
```

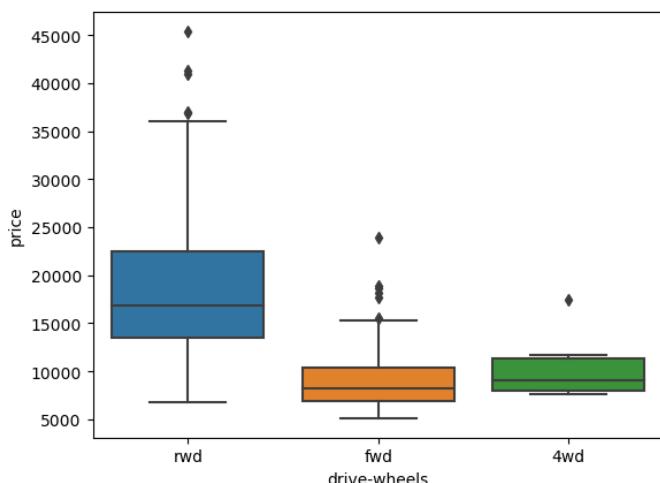


Здесь мы видим, что распределение цены между этими двумя категориями расположения двигателя, передним и задним, достаточно различается, чтобы рассматривать расположение двигателя в качестве потенциально хорошего предиктора цены.

```
# drive-wheels
```

```
sns.boxplot(x="drive-wheels", y="price", data=df)
```

```
:AxesSubplot:xlabel='drive-wheels', ylabel='price'>
```



Здесь мы видим, что распределение цены между разными категориями ведущих колес различается. Таким образом, ведущие колеса потенциально могут быть предиктором цены.

ОПИСАТЕЛЬНЫЙ СТАТИСТИЧЕСКИЙ АНАЛИЗ

Сначала рассмотрим переменные с помощью метода описания.

Функция `describe` автоматически вычисляет базовую статистику для всех непрерывных переменных. Любые значения `NaN` автоматически пропускаются в этой статистике.

Это покажет:

- количество для данной переменной
- среднее значение
- стандартное отклонение (std)
- минимальное значение
- IQR (интерквартильный размах: 25%, 50% и 75%)
- максимальное значение

Мы можем применить метод "описать" следующим образом:

`df.describe()`

	symboling	normalized-losses	wheel-base	length	width	height	curb-weight	engine-size	bore	stroke
count	201.000000	201.000000	201.000000	201.000000	201.000000	201.000000	201.000000	201.000000	201.000000	197.000000
mean	0.840796	122.000000	98.797015	0.837102	0.915126	53.766667	2555.666667	126.875622	3.330692	3.256904
std	1.254802	31.99625	6.066366	0.059213	0.029187	2.447822	517.296727	41.546834	0.268072	0.319256
min	-2.000000	65.00000	86.600000	0.678039	0.837500	47.800000	1488.000000	61.000000	2.540000	2.070000
25%	0.000000	101.000000	94.500000	0.801538	0.890278	52.000000	2169.000000	98.000000	3.150000	3.110000
50%	1.000000	122.000000	97.000000	0.832292	0.909722	54.100000	2414.000000	120.000000	3.310000	3.290000
75%	2.000000	137.000000	102.400000	0.881788	0.925000	55.500000	2926.000000	141.000000	3.580000	3.410000
max	3.000000	256.000000	120.900000	1.000000	1.000000	59.800000	4066.000000	326.000000	3.940000	4.170000

Значение по умолчанию «`describe`» пропускает переменные типа `object`. Мы можем применить метод «`describe`» к переменным типа «`object`» **следующим образом:**

`df.describe(include=['object'])`

	make	aspiration	num-of-doors	body-style	drive-wheels	engine-location	engine-type	num-of-cylinders	fuel-system	horsepower-binned
count	201	201	201	201	201	201	201	201	201	200
unique	22	2	2	5	3	2	6	7	8	3
top	toyota	std	four	sedan	fwd	front	ohc	four	mpfi	Low
freq	32	165	115	94	118	198	145	157	92	115

ПОДСЧЕТЫ ЗНАЧЕНИЙ

Подсчет значений - это хороший способ понять, сколько единиц каждой характеристики/переменной у нас есть. Мы можем применить метод "value_counts" к столбцу "drive-wheels". Не забывайте, что метод "value_counts" работает только с сериями pandas, а не с рамками данных pandas. В результате мы включаем только одну скобку `df['drive-wheels']`, а не две скобки `df[['drive-wheels']]`.

```
df['drive-wheels'].value_counts()
```

```
fwd    118  
rwd     75  
4wd      8  
Name: drive-wheels, dtype: int64
```

Можно конвертировать это в дата фрейм

```
df['drive-wheels'].value_counts().to_frame()
```

drive-wheels	
fwd	118
rwd	75
4wd	8

```
drive_wheels_counts = df['drive-wheels'].value_counts().to_frame()
```

```
drive_wheels_counts.rename(columns={'drive-wheels': 'value_counts'}, inplace=True)
```

```
drive_wheels_counts
```

value_counts	
fwd	118
rwd	75
4wd	8

```
drive_wheels_counts.index.name = 'drive-wheels'
```

```
drive_wheels_counts
```

value_counts	
drive-wheels	
fwd	118
rwd	75
4wd	8

Мы можем повторить описанный выше процесс для переменной «engine-location».

```
engine_loc_counts = df['engine-location'].value_counts().to_frame()
```

```
engine_loc_counts.rename(columns={'engine-location': 'value_counts'}, inplace=True)

engine_loc_counts.index.name = 'engine-location'

engine_loc_counts.head(10)
```

engine-location	value_counts
front	198
rear	3

Изучив подсчеты значений расположения двигателя, мы видим, что расположение двигателя не является хорошей предикторной переменной для цены. **Это связано с тем, что у нас есть только три автомобиля с задним расположением двигателя и 198 автомобилей с двигателем спереди, поэтому результат искажен.** Таким образом, мы не можем сделать никаких выводов относительно расположения двигателя.

ОСНОВЫ ГРУППИРОВКИ

Метод "groupby" группирует данные по различным категориям. Данные группируются на основе одной или нескольких переменных, и анализ проводится по отдельным группам.

Например, давайте сгруппируем данные по переменной "приводные колеса". Мы видим, что существует 3 различные категории приводных колес

```
df['drive-wheels'].unique()
```

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

Если мы хотим узнать в среднем, какой тип ведущего колеса является наиболее ценным, мы можем сгруппировать «ведущие колеса» и затем усреднить их. Мы можем выбрать столбцы «диски», «стиль кузова» и «цена», а затем присвоить их переменной «df_group_one».

```
df_group_one = df[['drive-wheels','body-style','price']]
```

Затем мы можем рассчитать среднюю цену для каждой из различных категорий данных.

grouping results

```
df_group_one = df_group_one.groupby(['drive-wheels'],as_index=False).mean()

df_group_one
```

	drive-wheels	price
0	4wd	10241.000000
1	fwd	9244.779661
2	rwd	19757.613333

Из наших данных следует, что заднеприводные автомобили в среднем самые дорогие, а полноприводные и переднеприводные примерно одинаковы по цене.

Вы также можете группировать по нескольким переменным. Например, давайте сгруппируем данные по "ведущим колесам" и "стилю кузова". Это сгруппирует кадр данных по уникальной комбинации 'drive-wheels' и 'body-style'. Мы можем сохранить результаты в переменной 'grouped_test1'.

grouping results

```
df_gptest = df[['drive-wheels','body-style','price']]  
  
grouped_test1 = df_gptest.groupby(['drive-wheels','body-style'],as_index=False).mean()  
  
grouped_test1
```

	drive-wheels	body-style	price
0	4wd	hatchback	7603.000000
1	4wd	sedan	12647.333333
2	4wd	wagon	9095.750000
3	fwd	convertible	11595.000000
4	fwd	hardtop	8249.000000
5	fwd	hatchback	8396.387755
6	fwd	sedan	9811.800000
7	fwd	wagon	9997.333333
8	rwd	convertible	23949.600000
9	rwd	hardtop	24202.714286
10	rwd	hatchback	14337.777778
11	rwd	sedan	21711.833333
12	rwd	wagon	16994.222222

Эти сгруппированные данные гораздо легче визуализировать, если их преобразовать в разворотную таблицу. Разворотная таблица похожа на электронную таблицу Excel, в которой одна переменная находится в столбце, а другая - в строке. Мы можем преобразовать кадр данных в сводную таблицу, используя метод "pivot" для создания сводной таблицы из групп.

В данном случае мы оставим переменную drive-wheels в качестве строк таблицы, а переменную pivot body-style - в качестве столбцов таблицы:

```
grouped_pivot = grouped_test1.pivot(index='drive-wheels',columns='body-style')
```

```
grouped_pivot
```

body-style	convertible	hardtop	hatchback	sedan	wagon
drive-wheels					
4wd	NaN	NaN	7603.000000	12647.333333	9095.750000
fwd	11595.0	8249.000000	8396.387755	9811.800000	9997.333333
rwd	23949.6	24202.714286	14337.777778	21711.833333	16994.222222

Часто у нас нет данных для некоторых поворотных ячеек. Мы можем заполнить эти отсутствующие ячейки значением 0, но потенциально можно использовать и любое другое значение. Следует отметить, что недостающие данные - довольно сложная тема, и это целый отдельный курс.

```
grouped_pivot = grouped_pivot.fillna(0) #fill missing values with 0
```

```
grouped_pivot
```

body-style	convertible	hardtop	hatchback	sedan	wagon
drive-wheels					
4wd	0.0	0.000000	7603.000000	12647.333333	9095.750000
fwd	11595.0	8249.000000	8396.387755	9811.800000	9997.333333
rwd	23949.6	24202.714286	14337.777778	21711.833333	16994.222222

Используйте функцию «groupby», чтобы найти среднюю «цену» каждого автомобиля на основе «стиля кузова».

```
df_gptest2 = df[['body-style','price']]
```

```
grouped_test_bodystyle = df_gptest2.groupby(['body-style'],as_index=False).mean()
```

```
grouped_test_bodystyle
```

	body-style	price
0	convertible	21890.500000
1	hardtop	22208.500000
2	hatchback	9957.441176
3	sedan	14459.755319
4	wagon	12371.960000

```
import matplotlib.pyplot as plt
```

```
%matplotlib inline
```

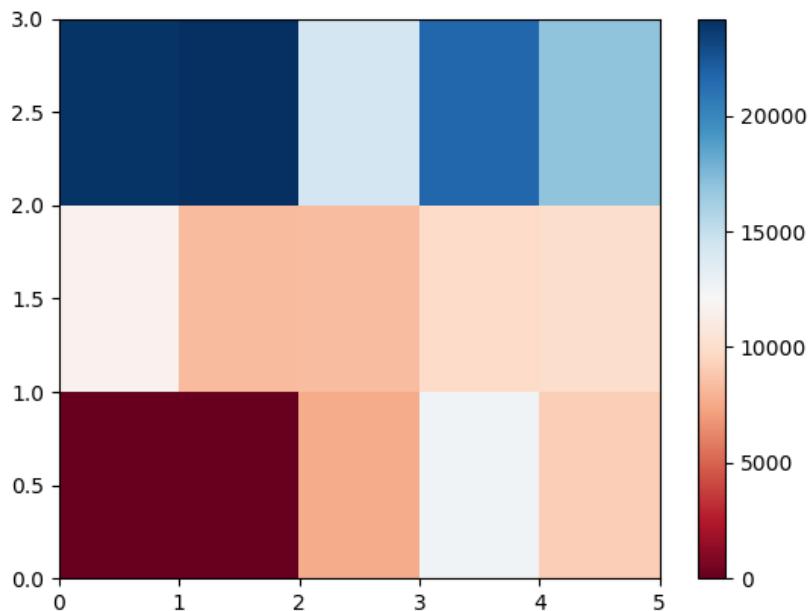
Давайте используем тепловую карту, чтобы визуализировать взаимосвязь между стилем кузова и ценой.

```
#use the grouped results
```

```
plt.pcolor(grouped_pivot, cmap='RdBu')
```

```
plt.colorbar()
```

```
plt.show()
```



Тепловая карта отображает целевую переменную (цену) пропорционально цвету относительно переменных 'drive-wheel' и 'body-style' на вертикальной и горизонтальной оси, соответственно. Это позволяет нам визуализировать, как цена связана с "ведущим колесом" и "стилем кузова".

Метки по умолчанию не несут никакой полезной информации. Давайте изменим это:

```
fig, ax = plt.subplots()
```

```
im = ax.pcolor(grouped_pivot, cmap='RdBu')
```

```
#label names
```

```
row_labels = grouped_pivot.columns.levels[1]
```

```

col_labels = grouped_pivot.index

#move ticks and labels to the center
ax.set_xticks(np.arange(grouped_pivot.shape[1]) + 0.5, minor=False)
ax.set_yticks(np.arange(grouped_pivot.shape[0]) + 0.5, minor=False)

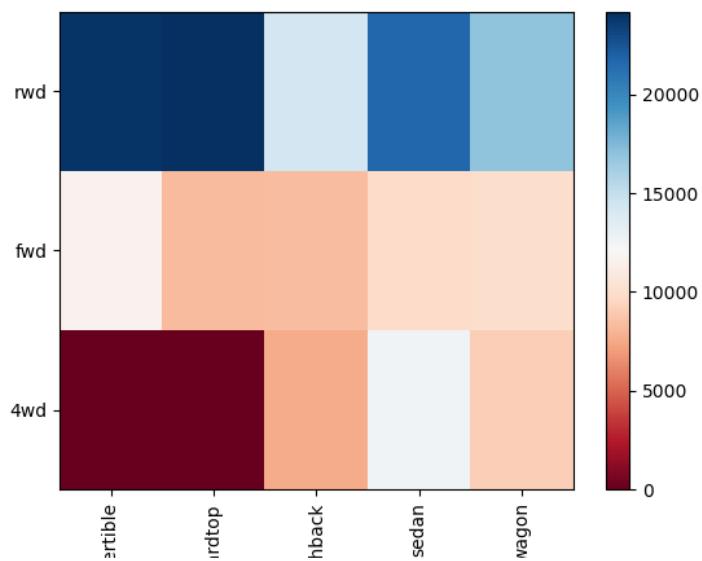
#insert labels
ax.set_xticklabels(row_labels, minor=False)
ax.set_yticklabels(col_labels, minor=False)

#rotate label if too long
plt.xticks(rotation=90)

fig.colorbar(im)

plt.show()

```



Визуализация очень важна в науке о данных, и пакеты визуализации Python предоставляют большую свободу. Более подробно мы рассмотрим их в отдельном курсе по визуализациям Python.

Главный вопрос, на который мы хотим ответить в этом модуле: "Какие основные характеристики оказывают наибольшее влияние на цену автомобиля?".

Чтобы лучше определить важные характеристики, мы посмотрим на корреляцию этих переменных с ценой автомобиля. Другими словами: как цена автомобиля зависит от этой переменной?

- Корреляция и причинно-следственная связь
- Корреляция: мера степени взаимозависимости между переменными.
- Причинность: связь между причиной и следствием между двумя переменными.

Важно знать разницу между этими двумя понятиями. Корреляция не подразумевает причинно-следственную связь. Определение корреляции намного проще, чем определение причинно-следственной связи, поскольку для определения причинно-следственной связи может потребоваться проведение независимых экспериментов.

Корреляция Пирсона

Корреляция Пирсона измеряет линейную зависимость между двумя переменными X и Y.

Результатирующий коэффициент представляет собой значение от -1 до 1 включительно, где:

1: идеальная положительная линейная корреляция.

0: Линейная корреляция отсутствует, две переменные, скорее всего, не влияют друг на друга.

-1: Идеальная отрицательная линейная корреляция.

Корреляция Пирсона - это метод по умолчанию функции "corr". Как и раньше, мы можем вычислить корреляцию Пирсона для переменных 'int64' или 'float64'.

df.corr()

	symboling	normalized-losses	wheel-base	length	width	height	curb-weight	engine-size	bore	stroke	comp
symboling	1.000000	0.466264	-0.535987	-0.365404	-0.242423	-0.550160	-0.233118	-0.110581	-0.140019	-0.008245	
normalized-losses	0.466264	1.000000	-0.056661	0.019424	0.086802	-0.373737	0.099404	0.112360	-0.029862	0.055563	
wheel-base	-0.535987	-0.056661	1.000000	0.876024	0.814507	0.590742	0.782097	0.572027	0.493244	0.158502	
length	-0.365404	0.019424	0.876024	1.000000	0.857170	0.492063	0.880665	0.685025	0.608971	0.124139	
width	-0.242423	0.086802	0.814507	0.857170	1.000000	0.306002	0.866201	0.729436	0.544885	0.188829	
height	-0.550160	-0.373737	0.590742	0.492063	0.306002	1.000000	0.307581	0.074694	0.180449	-0.062704	
curb-weight	-0.233118	0.099404	0.782097	0.880665	0.866201	0.307581	1.000000	0.849072	0.644060	0.167562	
engine-size	-0.110581	0.112360	0.572027	0.685025	0.729436	0.074694	0.849072	1.000000	0.572609	0.209523	
bore	-0.140019	-0.029862	0.493244	0.608971	0.544885	0.180449	0.644060	0.572609	1.000000	-0.055390	
stroke	-0.008245	0.055563	0.158502	0.124139	0.188829	-0.062704	0.167562	0.209523	-0.055390	1.000000	
compression-ratio	-0.182196	-0.114713	0.250313	0.159733	0.189867	0.259737	0.156433	0.028889	0.001263	0.187923	
horsepower	0.075819	0.217299	0.371147	0.579821	0.615077	-0.087027	0.757976	0.822676	0.566936	0.098462	
peak-rpm	0.279740	0.239543	-0.360305	-0.285970	-0.245800	-0.309974	-0.279361	-0.256733	-0.267392	-0.065713	
city-mpg	-0.035527	-0.225016	-0.470606	-0.665192	-0.633531	-0.049800	-0.749543	-0.650546	-0.582027	-0.034696	
highway-mpg	0.036233	-0.181877	-0.543304	-0.698142	-0.680635	-0.104812	-0.794889	-0.679571	-0.591309	-0.035201	
price	-0.082391	0.133999	0.584642	0.690628	0.751265	0.135486	0.834415	0.872335	0.543155	0.082310	
city-L/100km	0.066171	0.238567	0.476153	0.657373	0.673363	0.003811	0.785353	0.745059	0.554610	0.037300	
diesel	-0.196735	-0.101546	0.307237	0.211187	0.244356	0.281578	0.221046	0.070779	0.054458	0.241303	
cas	0.196735	0.101546	-0.307237	-0.211187	-0.244356	-0.281578	-0.221046	-0.070779	-0.054458	-0.241303	

Иногда мы хотим узнать значимость оценки корреляции.

P-значение

Что такое P-значение? **P-значение - это значение вероятности того, что корреляция между этими двумя переменными статистически значима.**

Обычно мы выбираем уровень значимости 0,05, что означает, что мы на 95% уверены в том, что корреляция между переменными значима.

По общему правилу, когда

p-значение < 0,001: мы говорим, что есть сильные доказательства того, что корреляция значима.

p-значение < 0,05: есть умеренные доказательства того, что корреляция значима.

p-значение < 0,1: есть слабые доказательства того, что корреляция значима.

Мы можем получить эту информацию, используя модуль "stats" в библиотеке "scipy".

```
# Импортируем статистическую библиотеку
```

```
from scipy import stats
```

Колесная база и цена

```
pearson_coef, p_value = stats.pearsonr(df['wheel-base'], df['price'])
```

```
print("The Pearson Correlation Coefficient is", pearson_coef, "with a P-value of P =", p_value)
```

OUTPUT: Коэффициент корреляции Пирсона составляет 0,5846418222655085 с Р-значением Р = 8,076488270732243e-20.

Заключение: Поскольку р-значение < < 0,001 корреляция между колесной базой и ценой статистически значима, хотя линейная зависимость не очень сильная (~0,585).

Мощность против цены

```
pearson_coef, p_value = stats.pearsonr(df['horsepower'], df['price'])

print("The Pearson Correlation Coefficient is", pearson_coef, " with a P-value of P = ",
p_value)
```

Коэффициент корреляции Пирсона равен 0,8095745670036559 с Р-значением Р = 6,369057428260101e-48.

Поскольку р-значение < < 0,001, корреляция между мощностью и ценой статистически значима, а линейная зависимость довольно сильная (~0,809, близко к 1).

Длина против цены

```
pearson_coef, p_value = stats.pearsonr(df['length'], df['price'])

print("The Pearson Correlation Coefficient is", pearson_coef, " with a P-value of P = ",
p_value)
```

Коэффициент корреляции Пирсона составляет 0,6906283804483643 с Р-значением Р = 8,01647746615853e-30.

Поскольку р-значение < 0,001, корреляция между длиной и ценой статистически значима, а линейная зависимость умеренно сильная (~0,691).

Ширина против цены

```
pearson_coef, p_value = stats.pearsonr(df['width'], df['price'])

print("The Pearson Correlation Coefficient is", pearson_coef, " with a P-value of P = ",
p_value )
```

Коэффициент корреляции Пирсона составляет 0,7512653440522666 с Р-значением Р = 9,200335510483739e-38.

Заключение: Поскольку р-значение <0,001, корреляция между шириной и ценой статистически значима, а линейная зависимость довольно сильная (~0,751).

ANOVA

ANOVA: дисперсионный анализ

Дисперсионный анализ (ANOVA) - это статистический метод, используемый для проверки наличия существенных различий между средними значениями двух или более групп. ANOVA возвращает два параметра:

F-тестовый показатель: ANOVA предполагает, что средние всех групп одинаковы, рассчитывает, насколько фактические средние отклоняются от предположения, и сообщает об этом в виде показателя F-теста. Большой показатель означает, что разница между средними больше.

P-значение: P-значение показывает, насколько статистически значимым является рассчитанное нами значение балла.

Если наша ценовая переменная сильно коррелирует с анализируемой переменной, мы ожидаем, что ANOVA даст значительный результат F-теста и небольшое p-значение.

Приводные колеса

Поскольку ANOVA анализирует разницу между различными группами одной и той же переменной, пригодится функция groupby. Поскольку алгоритм ANOVA усредняет данные автоматически, нам не нужно предварительно брать среднее значение.

Чтобы узнать, влияют ли различные типы "ведущих колес" на "цену", мы сгруппируем данные.

```
grouped_test2=df_gptest[['drive-wheels', 'price']].groupby(['drive-wheels'])
```

```
grouped_test2.head(2)
```

	drive-wheels	price
0	rwd	13495.0
1	rwd	16500.0
3	fwd	13950.0
4	4wd	17450.0
5	fwd	15250.0
136	4wd	7603.0

```
df_gptest
```

	drive-wheels	body-style	price
0	rwd	convertible	13495.0
1	rwd	convertible	16500.0
2	rwd	hatchback	16500.0
3	fwd	sedan	13950.0
4	4wd	sedan	17450.0
...
196	rwd	sedan	16845.0
197	rwd	sedan	19045.0
198	rwd	sedan	21485.0
199	rwd	sedan	22470.0
200	rwd	sedan	22625.0

Мы можем получить значения группы методов, используя метод «get_group».

```
grouped_test2.get_group('4wd')['price']
```

```
4      17450.0
136    7603.0
140    9233.0
141    11259.0
144    8013.0
145    11694.0
150    7898.0
151    8778.0
Name: price, dtype: float64
```

Мы можем использовать функцию «**f_oneway**» в модуле «Статистика», чтобы получить оценку F-теста и P-значение.

ANOVA

```
f_val, p_val = stats.f_oneway(grouped_test2.get_group('fwd')['price'],
grouped_test2.get_group('rwd')['price'], grouped_test2.get_group('4wd')['price'])

print("ANOVA results: F=", f_val, ", P =", p_val)
```

ВЫВОД: ANOVA results: F= 67.95406500780399 , P = 3.3945443577151245e-23

Это отличный результат с большим значением F-теста, показывающим сильную корреляцию, и P-значением почти 0, означающим почти уверенную статистическую значимость. Но означает ли это, что все три тестируемые группы настолько сильно коррелируют между собой?

Давайте рассмотрим их по отдельности.

fwd and rwd

```
f_val, p_val = stats.f_oneway(grouped_test2.get_group('fwd')['price'],  
grouped_test2.get_group('rwd')['price'])
```

```
print( "ANOVA results: F=", f_val, ", P =", p_val )
```

Вывод: ANOVA results: F= 130.5533160959111 , P = 2.2355306355677845e-23

4wd and rwd

```
f_val, p_val = stats.f_oneway(grouped_test2.get_group('4wd')['price'],  
grouped_test2.get_group('rwd')['price'])
```

```
print( "ANOVA results: F=", f_val, ", P =", p_val )
```

Вывод: ANOVA results: F= 8.580681368924756 , P = 0.004411492211225333

4wd and fwd

```
f_val, p_val = stats.f_oneway(grouped_test2.get_group('4wd')['price'],  
grouped_test2.get_group('fwd')['price'])
```

```
print("ANOVA results: F=", f_val, ", P =", p_val)
```

Вывод: ANOVA results: F= 0.665465750252303 , P = 0.41620116697845655

Заключение: Важные переменные

Теперь мы лучше представляем себе, как выглядят наши данные и какие переменные важно учитывать при прогнозировании цены автомобиля. Мы сузили список до следующих переменных:

Непрерывные числовые переменные:

- Длина
- Ширина
- Снаряженная масса
- Размер двигателя
- Лошадиная сила
- Расход по городу
- Расход по шоссе
- Колесная база
- Диаметр

Категориальные переменные:

- Приводные колеса

Поскольку мы переходим к созданию моделей машинного обучения для автоматизации нашего анализа, добавление в модель переменных, которые значимо влияют на нашу целевую переменную, улучшит эффективность прогнозирования нашей модели.

В этом видео мы рассмотрим разработку модели, пытаясь предсказать цену автомобиля с помощью нашего набора данных.

В этом модуле вы узнаете о **простой и множественной линейной регрессии, оценке моделей с помощью визуализации, полиномиальной регрессии и конвейерах, R-квадрат и MSE для оценки на выборке, прогнозирования и принятия решений, и как определить справедливую стоимость подержанного автомобиля.**

Модель или оценочный показатель можно представить как математическое уравнение, используемое для прогнозирования значения при наличии одного или нескольких других значений.

Соотношение одной или нескольких независимых переменных или характеристик с зависимыми переменными.

Например, вы вводите в модель автомобиля показатель миль на галлон по шоссе в качестве независимой переменной или характеристики, выходом модели или зависимой переменной является цена.

Обычно, чем больше у вас соответствующих данных, тем точнее будет ваша модель. **Например, вы вводите в модель несколько независимых переменных или характеристик.**

Таким образом, ваша модель может предсказать более точную цену на автомобиль. Чтобы понять, почему важно больше данных, рассмотрим следующую ситуацию.

У вас есть два почти одинаковых автомобиля.

Розовые автомобили продаются значительно дешевле. Вы хотите использовать свою модель для определения цены двух автомобилей, одного розового, другого красного. Если независимые переменные или характеристики вашей модели не включают цвет, ваша модель предскажет одинаковую цену для автомобилей, которые могут продаваться гораздо дешевле.

В дополнение к получению большего количества данных, вы можете попробовать различные типы моделей.

В этом курсе вы узнаете о простой линейной регрессии, множественной линейной регрессии и полиномиальной регрессии.

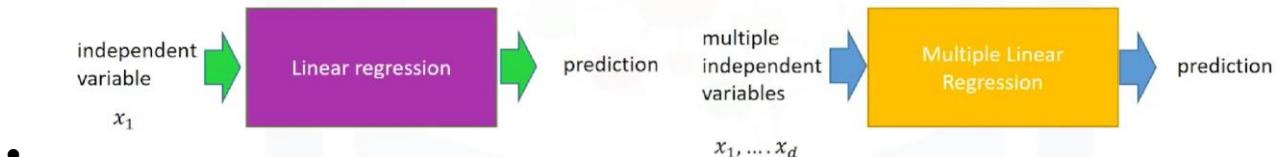
РЕГРЕССИИ

В этом видео мы поговорим о простой линейной регрессии и множественной линейной регрессии.

- **Линейная регрессия относится к одной независимой переменной для составления прогноза.**
- **Множественная линейная регрессия относится к нескольких независимых переменных для составления прогноза.**
- **Простая линейная регрессия или SLR – это метод, помогающий понять взаимосвязь между двумя переменными. Предсказывающая независимая переменная x и целевая зависимая переменная y.**

Introduction

- Linear regression will refer to one independent variable to make a prediction
- Multiple Linear Regression will refer to multiple independent variables to make a prediction



Мы хотели бы прийти к линейной зависимости между переменными, показанными здесь.

Параметр b_0 - это перехват.

Параметр b_1 - это наклон.

Simple Linear Regression

1. The predictor (independent) variable - x
2. The target (dependent) variable - y

$$y = b_0 + b_1 x$$

- b_0 : the intercept
- b_1 : the slope

Когда мы подгоним или обучим модель, мы получим эти параметры.

Этот шаг требует много математических вычислений, поэтому мы не будем останавливаться на этой части.

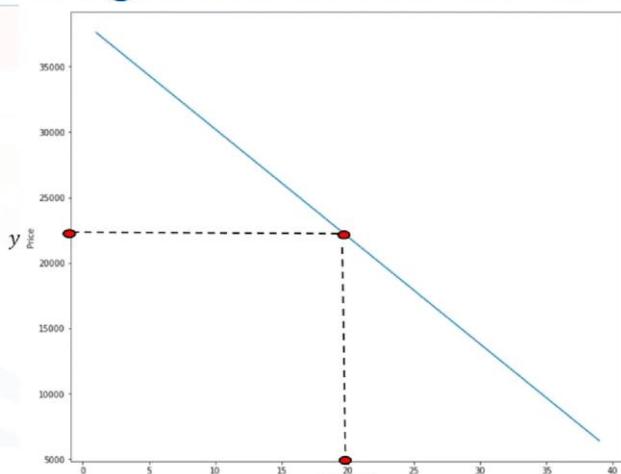
Давайте уточним шаг прогнозирования. Трудно определить, сколько стоит автомобиль, но в руководстве пользователя есть данные о расходе топлива на галлон по шоссе.

Если мы предположим, что между этими переменными существует линейная зависимость, мы можем использовать эту связь для формулировки модели, чтобы определить цену автомобиля.

Если пробег на галлон по шоссе составляет 20 миль, мы можем ввести это значение в модель, чтобы получить прогноз в \$22 000.

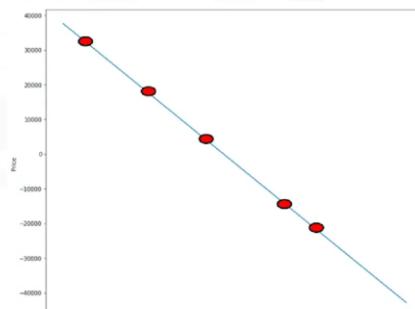
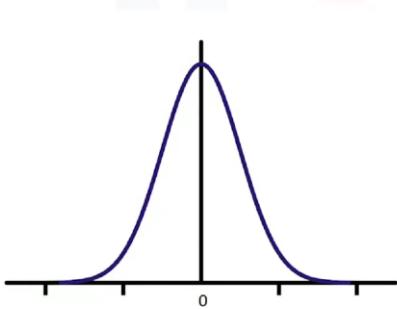
Simple Linear Regression: Prediction

$$\begin{aligned}y &= 38423 - 821x \\&= 38423 - 821(20) \\&= 22\,003\end{aligned}$$



Для того чтобы определить линию, мы берем точки данных из нашего набора данных, отмеченные здесь красным цветом.

Simple Linear Regression: Fit

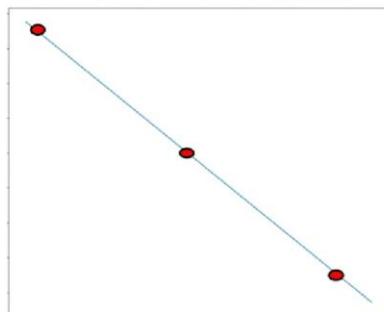


Затем мы используем эти тренировочные точки для подгонки нашей модели. Результаты обучения точек - это параметры. Обычно мы храним точки данных в массивах data frame или память.

Значение, которое мы хотим предсказать, называется целевой величиной, которую мы храним в массиве y . Независимую переменную мы храним в кадре данных или массиве x . Каждая выборка соответствует отдельной строке в каждом фрейме данных или массиве.

Во многих случаях на то, сколько люди платят за автомобиль, влияет множество факторов. Например, марка или возраст автомобиля.

Simple Linear Regression: Fit

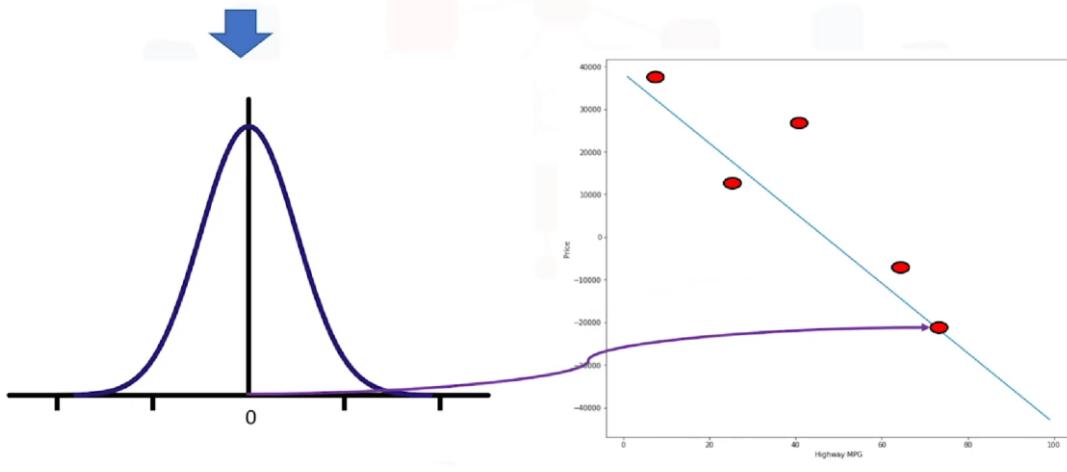


$$X = \begin{bmatrix} 0 \\ 20 \\ 40 \end{bmatrix} \quad Y = \begin{bmatrix} 38423 \\ 22003 \\ 5583 \end{bmatrix}$$

В данной модели эта неопределенность учитывается путем предполагая, что к точке на линии добавляется небольшая случайная величина. **Это называется шумом.**

На рисунке слева показано распределение шума. Вертикальная ось показывает добавленную величину, а горизонтальная ось показывает вероятность того, что значение будет добавлено.

Simple Linear Regression: Fit

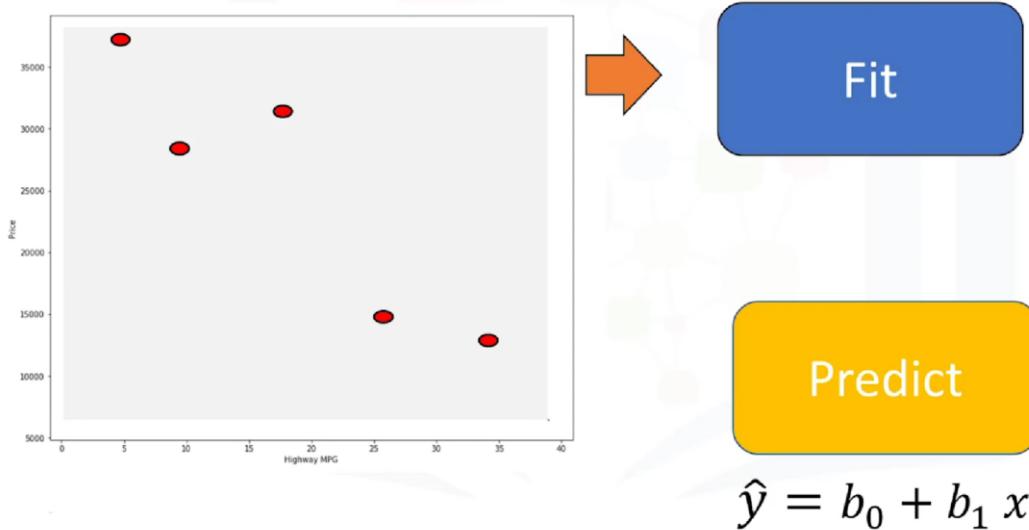


Обычно добавляется небольшое положительное значение или небольшое отрицательное значение. Иногда добавляются большие значения.

Но в большинстве случаев добавленные значения близки к нулю. Мы можем обобщить этот процесс следующим образом. У нас есть набор обучающих точек.

Мы используем эти обучающие точки для подгонки или обучения модели и получения параметров. Затем мы используем эти параметры в модели. Теперь у нас есть модель. Мы используем \hat{y} на y , чтобы обозначить модель как оценку.

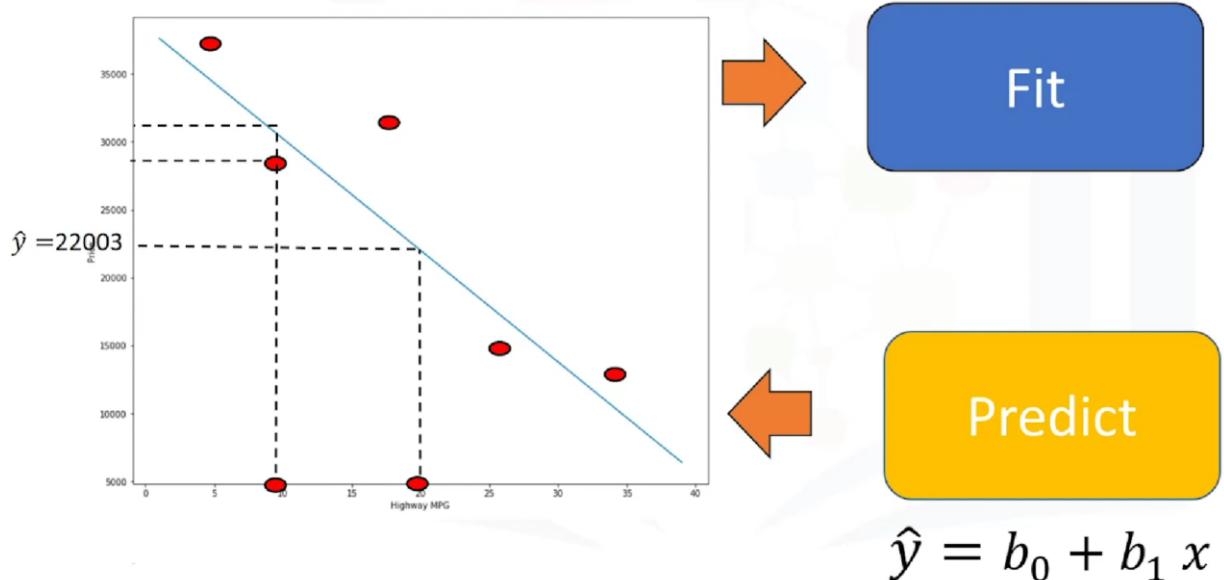
Simple Linear Regression



Мы можем использовать эту модель для прогнозирования значений, которые мы не видели. Например, у нас нет автомобиля с расходом 20 миль на галлон.

Мы можем использовать нашу модель для прогнозирования цены этого автомобиля. Но не забывайте, что наша модель не всегда верна.

Simple Linear Regression



Мы можем убедиться в этом, сравнив прогнозируемое значение с фактическим. У нас есть образец для 10 миль на галлон, но прогнозируемое значение не совпадает с фактическим. Если линейное предположение верно, эта ошибка вызвана шумом.

- Но могут быть и другие причины. Чтобы подогнать модель в Python, **сначала мы импортируем линейную модель из sklearn, затем создаем объект линейной регрессии с помощью конструктора.**

Fitting a Simple Linear Model Estimator

- X :Predictor variable
- Y: Target variable

1. Import linear_model from scikit-learn

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

2. Create a Linear Regression Object using the constructor :

```
lm=LinearRegression()
```

- Мы определяем переменную-предсказатель и целевую переменную, затем используем метод fit для подгонки модели и нахождения параметров b_0 и b_1 .

Fitting a Simple Linear Model

- We define the predictor variable and target variable

```
X = df[['highway-mpg']]  
Y = df['price']
```

- Then use lm.fit (X, Y) to fit the model , i.e fine the parameters b_0 and b_1

```
lm.fit(X, Y)
```

- We can obtain a prediction

```
Yhat=lm.predict(X)
```

Входными данными являются признаки и цели. Мы можем получить предсказание с помощью метода predict. Выходом является массив.

Fitting a Simple Linear Model

- We define the predictor variable and target variable

```
X = df[['highway-mpg']]  
Y = df['price']
```

- Then use lm.fit (X, Y) to fit the model , i.e fine the parameters b_0 and b_1

```
lm.fit(X, Y)
```

- We can obtain a prediction

```
Yhat=lm.predict(X)
```

Yhat	X
2	5
:	
3	4

Массив имеет такое же количество выборок, как и входной x. Перехват b_0 является атрибутом объекта lm.

Наклон b_1 также является атрибутом объекта lm. Взаимосвязь между ценой и количеством миль на галлон дается этим уравнением, выделенным жирным шрифтом, цена равна 38 423,31 минус 821,73 миль на галлон по шоссе, как уравнение, которое мы обсуждали ранее.

SLR – Estimated Linear Model

- We can view the intercept (b_0): lm.intercept_
38423.305858

- We can also view the slope (b_1): lm.coef_
-821.73337832

- The Relationship between Price and Highway MPG is given by:

$$\text{Price} = 38423.31 - 821.73 * \text{highway-mpg}$$

$$\hat{Y} = b_0 + b_1 x$$

Множественная линейная регрессия используется для объяснения взаимосвязи между одной непрерывной целевой переменной y и двумя или более предикторными переменными x.

Если у нас есть, например, 4 предикторные переменные, то b_0 перехват x равен нулю b_1 коэффициент или параметр x_1 , b_2 коэффициент параметра x_2 и так далее.

Multiple Linear Regression (MLR)

$$\hat{Y} = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + b_4x_4$$

- b_0 : intercept ($X=0$)
- b_1 : the coefficient or parameter of x_1
- b_2 : the coefficient of parameter x_2 and so on..

Если есть только две переменные, то мы можем визуализировать значения.

Рассмотрим следующую функцию:

Переменные x_1 и x_2 можно визуализировать на двумерной плоскости.

Рассмотрим пример на следующем слайде. Таблица содержит различные значения предикторных переменных x_1 и x_2 .

Fitting a Multiple Linear Model Estimator

1. We can extract the for 4 predictor variables and store them in the variable Z

```
Z = df[['horsepower', 'curb-weight', 'engine-size', 'highway-mpg']]
```

2. Then train the model as before:

```
lm.fit(Z, df['price'])
```

3. We can also obtain a prediction

```
Yhat=lm.predict(X)
```

Положение каждой точки на двумерной плоскости выделено соответствующим цветом. Каждое значение предикторных переменных x_1 и x_2 будет отображено на новое значение y , y hat. Новые значения y y hat отображаются в вертикальном направлении с высотой, пропорциональной значению, которое принимает y hat.

Мы можем построить множественную линейную регрессию следующим образом.

Мы можем извлечь четыре предикторные переменные и сохранить их в переменной z , а затем обучить модель как и раньше, используя метод train with признаками или зависимыми переменными и целевыми двоеточиями.

Fitting a Multiple Linear Model Estimator

1. We can extract the four predictor variables and store them in the variable Z

```
Z = df[['horsepower', 'curb-weight', 'engine-size', 'highway-mpg']]
```

2. Then train the model as before:

```
lm.fit(Z, df['price'])
```

3. We can also obtain a prediction

```
Yhat=lm.predict(X)
```

The diagram illustrates a linear transformation process. On the left, there is a matrix labeled X with columns x_1, x_2, x_3, x_4 . An arrow points from this matrix to a vertical column labeled \hat{Y} with elements 2, :, 3. This represents the relationship $\hat{Y} = b_0 + b_1 x_1 + b_2 x_2 + b_3 x_3 + b_4 x_4$.

x_1	x_2	x_3	x_4	\hat{Y}
3	5	-4	3	2
:	:	:	:	:
2	4	2	-4	3

MLR – Estimated Linear Model

1. Find the intercept (b_0)

```
lm.intercept_
-15678.742628061467
```

2. Find the coefficients (b_1, b_2, b_3, b_4)

```
lm.coef_
array([52.65851272, 4.69878948, 81.95906216, 33.58258185])
```

The Estimated Linear Model:

- Price = -15678.74 + (52.66) * horsepower + (4.70) * curb-weight + (81.96) * engine-size + (33.58) * highway-mpg

Мы также можем получить предсказание, используя метод predict. В этом случае на вход подается массив или кадр данных с четырьмя столбцами. Количество строк соответствует количеству выборок. На выходе получается массив с тем же количеством элементов, что и количество выборок. Перехват является атрибутом объекта, а коэффициенты также являются атрибутами. Полезно визуализировать уравнение, заменив имена зависимых переменных на реальные имена.

Это идентично форме, которую мы обсуждали ранее.

ВИЗУАЛИЗАЦИЯ. Графики Регрессии

В этом видео мы рассмотрим оценку модели с помощью визуализации.

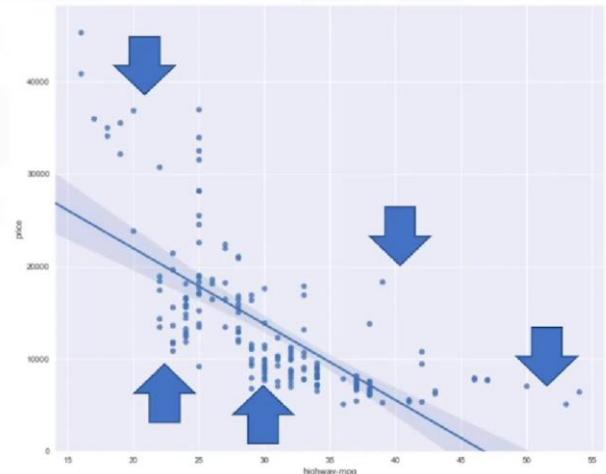
Графики регрессии являются хорошей оценкой взаимосвязи между двумя переменными, силу корреляции, и направление связи (положительное или отрицательное).

- Горизонтальная ось - независимая переменная.
- Вертикальная ось - зависимая переменная.

Regression Plot

Regression Plot shows us a combination of:

- The scatterplot: where each point represents a different y
- The fitted linear regression line (\hat{y})



Каждая точка представляет собой отдельную целевую точку. Подогнанная линия представляет собой прогнозируемое значение. Существует несколько способов построения графика регрессии.

Простой способ - использовать regplot из библиотеки seaborn. Сначала "import seaborn". Затем используйте функцию "regplot".

- Параметр x - это имя столбца, который содержит независимую переменную или признак.
- Параметр y - это имя столбца, который содержит имя зависимой переменной или цели.
- **Параметр data - это имя датафрейма. Результат выдается в виде графика.**

Regression Plot

```
import seaborn as sns
sns.regplot(x="highway-mpg", y="price", data=df)
plt.ylim(0,)
```

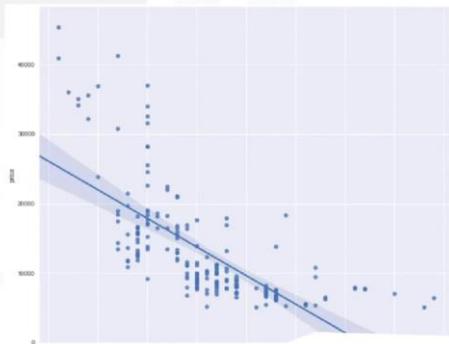


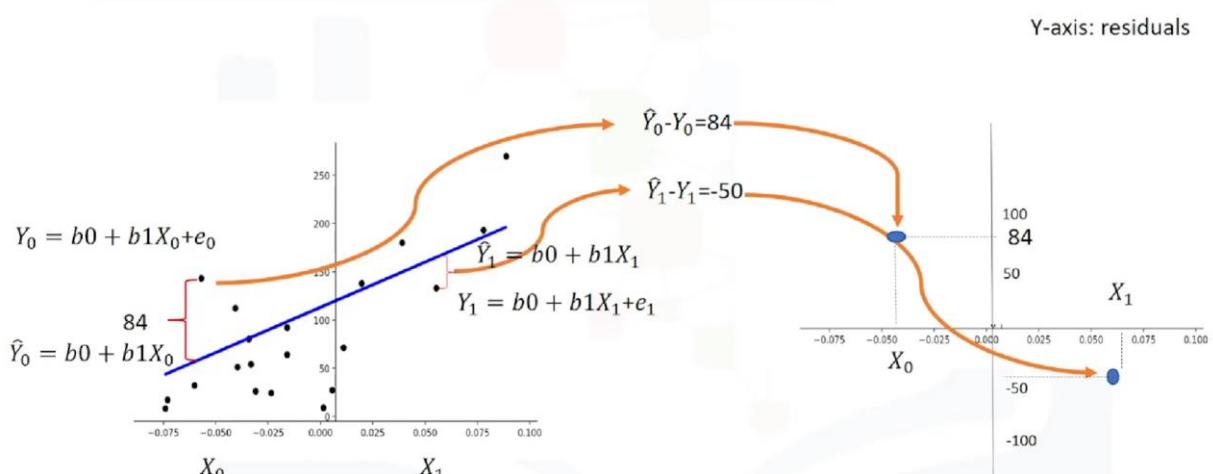
График остатков представляет собой ошибку между фактическим значением.

Рассматривая прогнозируемое и фактическое значение, мы видим разницу.

Мы получаем это значение путем вычитания предсказанного значения, и фактического значения цели. Затем мы откладываем это значение на вертикальной оси с независимой переменной в качестве горизонтальной оси.

Аналогично, для второй выборки мы повторяем процесс. Вычитаем целевое значение из прогнозируемого значения. Затем откладываем значение соответственно.

Residual Plot



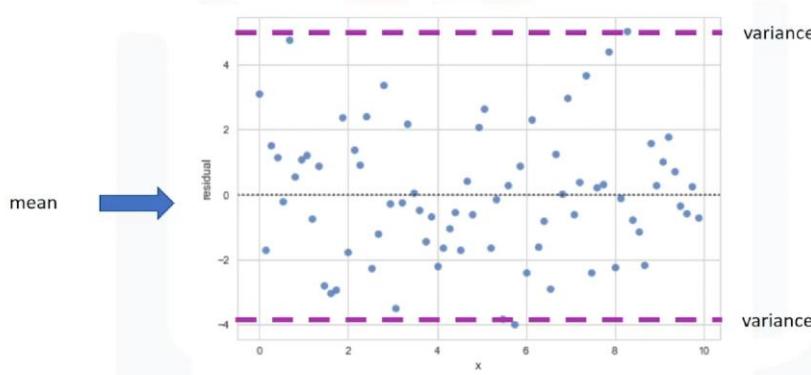
Взгляд на график дает нам некоторое представление о наших данных. Мы ожидаем, что результаты будут иметь нулевое среднее значение, равномерно распределены вокруг оси x с одинаковой дисперсией. Кривизна отсутствует.

Такой тип графика остатков предполагает использование линейного графика.

В этом графике остатков есть кривизна. Значения ошибки изменяются с ростом x.

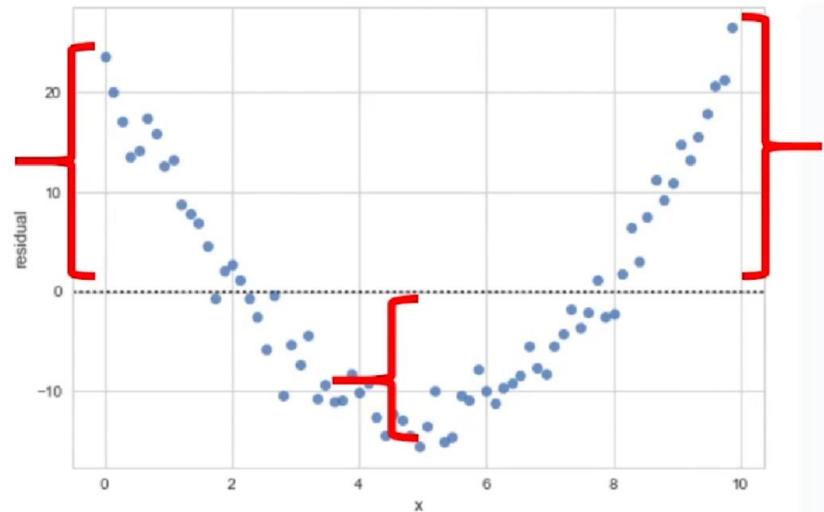
Например, в области, все остаточные ошибки положительны.

Residual Plot



В этой области остаточные ошибки отрицательны. В конечном месте ошибка велика. Остатки разделены неслучайно.

Residual Plot

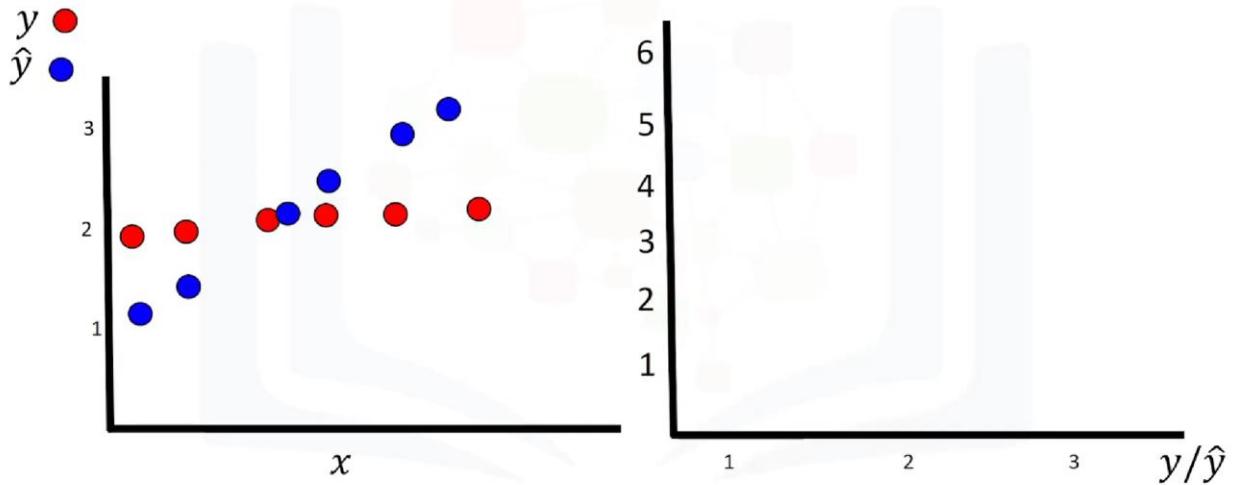


- Not randomly spread out around the x-axis

Это говорит о том, что линейное предположение неверно. Этот график говорит о нелинейной функции.

Мы разберемся с этим в следующем разделе. На этом графике мы видим, что дисперсия остатков увеличивается с ростом x.

Distribution Plots



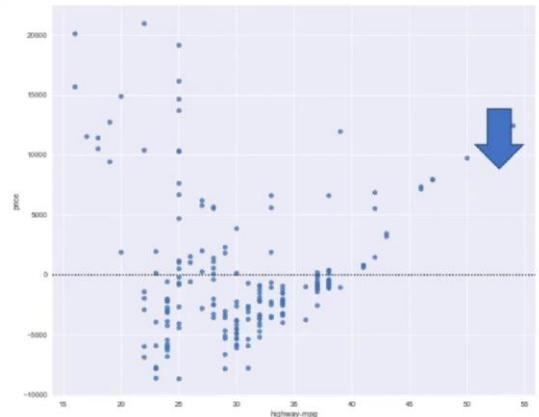
Следовательно, наша модель неверна.

Мы можем использовать программу seaborn для построения графика остатков.

- Сначала "импортируйте seaborn".
- Мы используем функцию "residplot".

Residual Plot

```
import seaborn as sns
sns.residplot(df['highway-mpg'], df['price'])
```



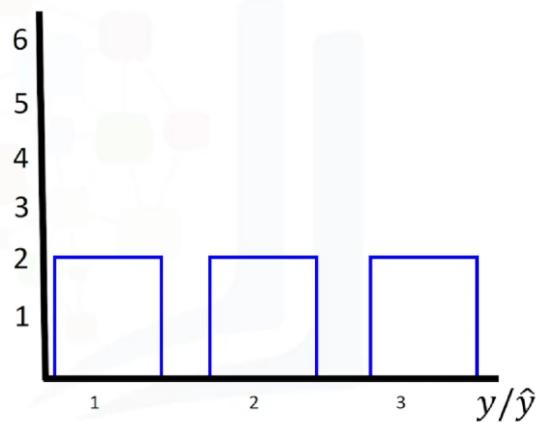
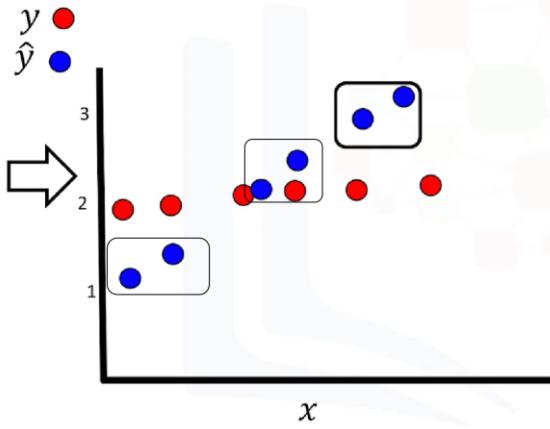
Первый параметр - это серия зависимых переменных или признаков. Второй параметр - серия зависимой переменной или цели. В данном случае мы видим, что остатки имеют кривизну. На графике распределения предсказанное значение сравнивается с фактическим.

Эти графики чрезвычайно полезны для визуализации моделей с более чем одной независимой переменной или характеристикой. Давайте рассмотрим упрощенный пример.

Мы рассмотрели вертикальную ось. Затем мы подсчитали и построили график количества предсказанных точек, которые приблизительно равны единице.

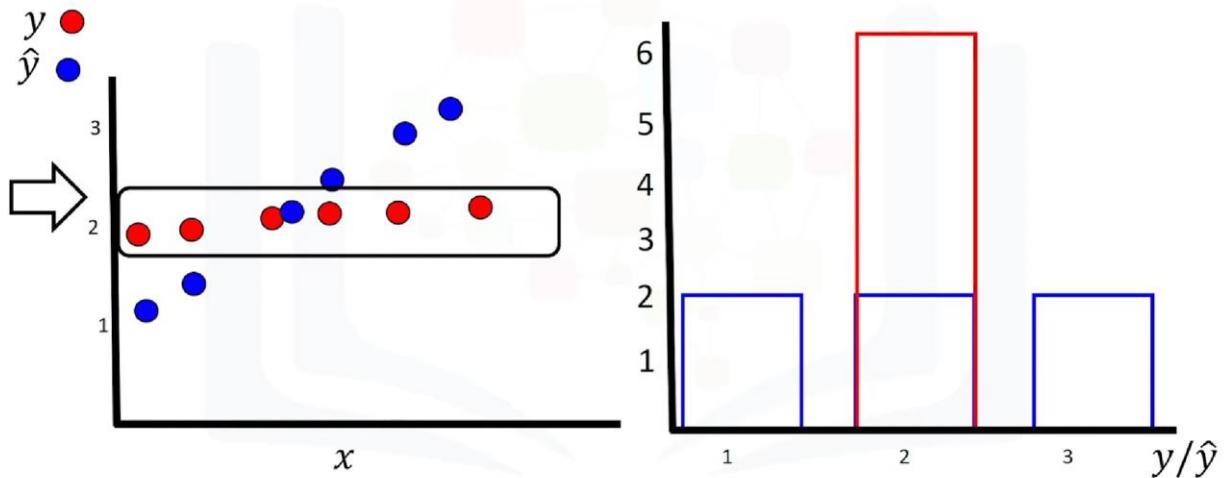
Затем мы подсчитываем и строим график количества предсказанных точек, которые приблизительно равны двум. Мы повторяем этот процесс.

Distribution Plots



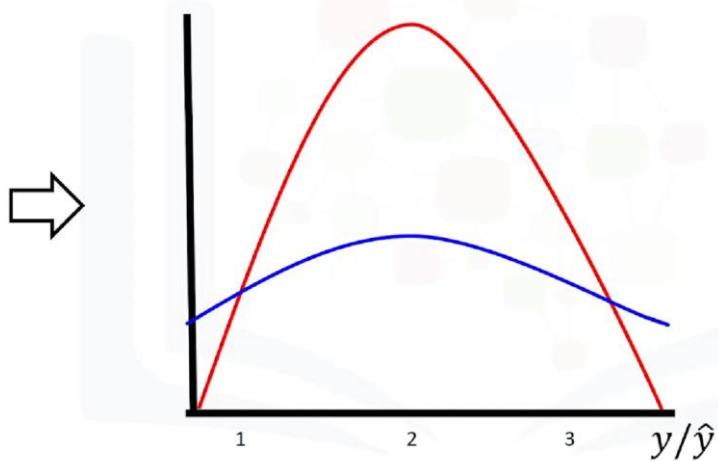
Для предсказанных точек они приблизительно равны трем. Затем мы повторяем процесс для целевых значений. В данном случае все целевые значения приблизительно равны двум. Значения целевых и прогнозируемых значений являются непрерывными.

Distribution Plots



Гистограмма предназначена для дискретных значений. Поэтому pandas преобразует их в распределение. Вертикальная ось масштабируется так, чтобы площадь под распределением была равна единице.

Distribution Plots

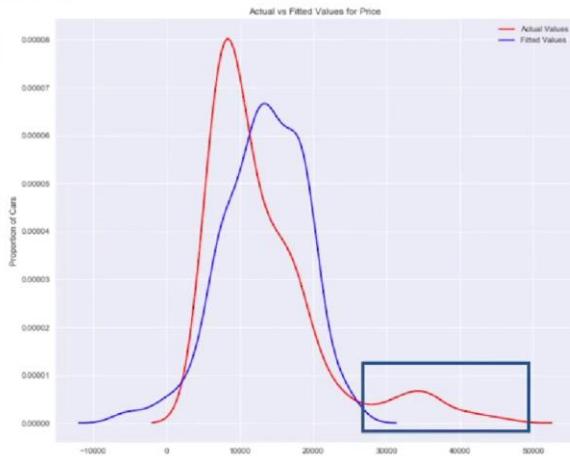


Это пример использования графика распределения. Зависимой переменной или характеристикой является цена. Подогнанные значения, полученные в результате модели, выделены синим цветом.

Distribution Plots

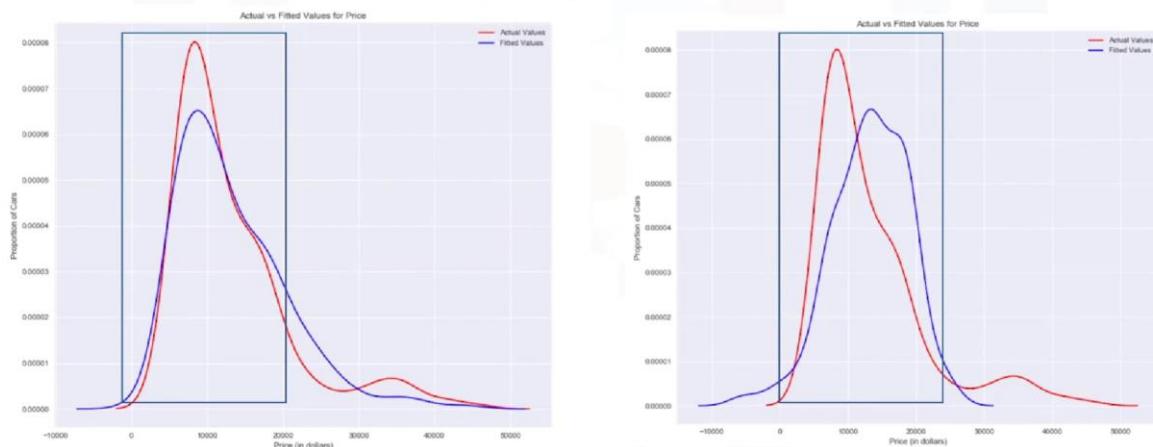
Compare the distribution plots:

- The fitted values that result from the model
- The actual values



Фактические значения - красным. Мы видим, что прогнозные значения для цен в диапазоне от 40 000 до 50 000 являются неточными.

MLR – Distribution Plots



Цены в области от 10 000 до 20 000 гораздо ближе к целевому значению. В этом примере мы используем несколько признаков или независимых переменных.

Сравнивая его с графиком на последнем слайде, мы видим, что предсказанные значения гораздо ближе к целевым. Вот код для создания графика распределения.

Distribution Plots

```
import seaborn as sns

ax1 = sns.distplot(df['price'], hist=False, color="r", label="Actual Value")

sns.distplot(Yhat, hist=False, color="b", label="Fitted Values" , ax=ax1)
```

В качестве параметра используются фактические значения. Мы хотели получить распределение, а не гистограмму. Поэтому мы хотим, чтобы параметры `hist` были установлены в `false`.

Цвет - красный. Метка также включена. Прогнозируемые значения включены для второго графика. Остальные параметры устанавливаются соответствующим образом.

ПОЛИНОМИАЛЬНАЯ РЕГРЕССИЯ.

В этом видео мы расскажем о полиномиальной регрессии и конвейерах.

Что делать, если линейная модель не подходит для наших данных?

Давайте рассмотрим другой тип регрессионной модели. Полиномиальная регрессия.

Мы преобразуем наши данные **в полином**, а затем используем линейную регрессию для подгонки параметра.

Затем мы обсудим конвейеры. **Конвейеры - это способ упростить ваш код.**

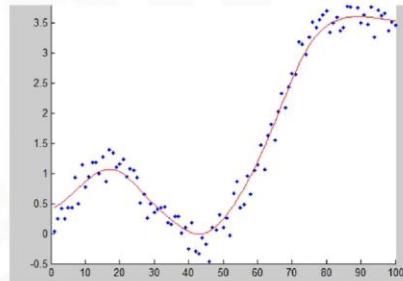
Полиномиальная регрессия - это частный случай общей линейной регрессии. Этот метод полезен для описания криволинейных зависимостей. Что такое криволинейная зависимость?

Polynomial Regressions

- A special case of the general linear regression model
- Useful for describing curvilinear relationships

Curvilinear relationships:

By squaring or setting higher-order terms
of the predictor variables



Это то, что вы получаете, возводя в квадрат или задавая условия более высокого порядка для предикторных переменных в модели, преобразующей данные.

Модель может быть квадратичной, что означает, что предикторная переменная в модели возводится в квадрат.

Polynomial Regression

- Quadratic – 2nd order

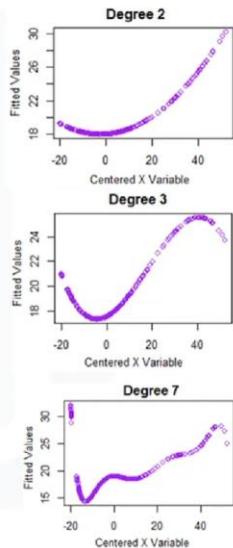
$$\hat{Y} = b_0 + b_1 x_1 + b_2 (x_1)^2$$

- Cubic – 3rd order

$$\hat{Y} = b_0 + b_1 x_1 + b_2 (x_1)^2 + b_3 (x_1)^3$$

- Higher order

$$\hat{Y} = b_0 + b_1 x_1 + b_2 (x_1)^2 + b_3 (x_1)^3 + \dots$$



Мы используем скобку, чтобы обозначить ее как экспоненту. Это полиномиальная регрессия второго порядка, с рисунком, изображающим функцию.

Модель может быть кубической, что означает, что переменная-предсказатель возведена в куб. Это полиномиальная регрессия третьего порядка.

Рассматривая рисунок, мы видим, что функция имеет большую вариацию. Существуют также полиномиальные регрессии более высокого порядка.

Когда хорошего соответствия не удалось достичь с помощью второго или третьего порядка. Мы можем видеть на рисунках, как сильно меняются графики, когда мы меняем порядок полиномиальной регрессии.

Степень регрессии имеет большую разницу и может привести к лучшему соответствию, если вы выберете правильное значение.

Во всех случаях связь между переменной и параметром всегда линейна.

Давайте рассмотрим пример на наших данных, где мы генерируем модель полиномиальной регрессии.

В Python мы делаем это с помощью функции polyfit.

В этом примере мы разрабатываем основу полиномиальной регрессионной модели третьего порядка. Мы можем распечатать модель.

Символическая форма для модели задается следующим выражением:

- отрицательное значение 1,557 x_1 в кубе плюс 204,8 x_1
- квадрат плюс 8965 x_1 плюс 1,37 умножить на 10 в степени

Polynomial Regression with More than One Dimension

- We can also have multi dimensional polynomial linear regression

$$\hat{Y} = b_0 + b_1 X_1 + b_2 X_2 + b_3 X_1 X_2 + b_4 (X_1)^2 + b_5 (X_2)^2 + ..$$

Мы также можем иметь многомерную полиномиальную линейную регрессию. Выражение может стать сложным.

Вот только некоторые из терминов для двумерного полинома второго порядка. Функция polyfit в Numpy не может выполнить этот тип регрессии.

Polynomial Regression with More than One Dimension

- The "preprocessing" library in scikit-learn,

```
from sklearn.preprocessing import PolynomialFeatures  
pr=PolynomialFeatures(degree=2, include_bias=False)
```

Мы используем библиотеку препроцессинга в scikit-learn для создания объекта полиномиальной функции. Конструктор принимает в качестве параметра степень полинома. Затем мы преобразуем признаки в полиномиальный признак с помощью метода fit_transform.

Давайте рассмотрим более интуитивный пример.

Рассмотрим признаки, показанные здесь.

Polynomial Regression with More than One Dimension

```
pr=PolynomialFeatures(degree=2)
```

X_1	X_2
1	2

```
pr=PolynomialFeatures(degree=2,include_bias=False)  
pr.fit_transform([[1,2]])
```



X_1	X_2	X_1X_2	X_1^2	X_2^2
1	2	(1) 2	1	(2) ²
1	2	2	1	4

Применив метод, мы преобразуем данные, теперь у нас есть новый набор признаков, которые являются преобразованной версией наших исходных признаков.

Поскольку размерность данных становится больше, мы можем захотеть нормализовать несколько признаков в scikit-learn.

Вместо этого мы можем использовать модуль предварительной обработки, чтобы упростить многие задачи. Например, мы можем нормировать каждый признак одновременно. Мы импортируем StandardScaler.

- Обучаем объект,
- подгоняем объект масштаба,

- затем преобразуем данные в новый кадр данных на массиве x_scale.

Pre-processing

- For example we can Normalize the each feature simultaneously

```
from sklearn.preprocessing import StandardScaler  
SCALE=StandardScaler()  
SCALE.fit(x_data[['horsepower', 'highway-mpg']])  
x_scale=SCALE.transform(x_data[['horsepower', 'highway-mpg']])
```

Существует больше методов нормализации, доступных в библиотека предварительной обработки, а также другие преобразования.

Мы можем упростить наш код, используя библиотеку конвейера. Для получения прогноза существует множество этапов. Например, нормализация, полиномиальное преобразование и линейная регрессия.

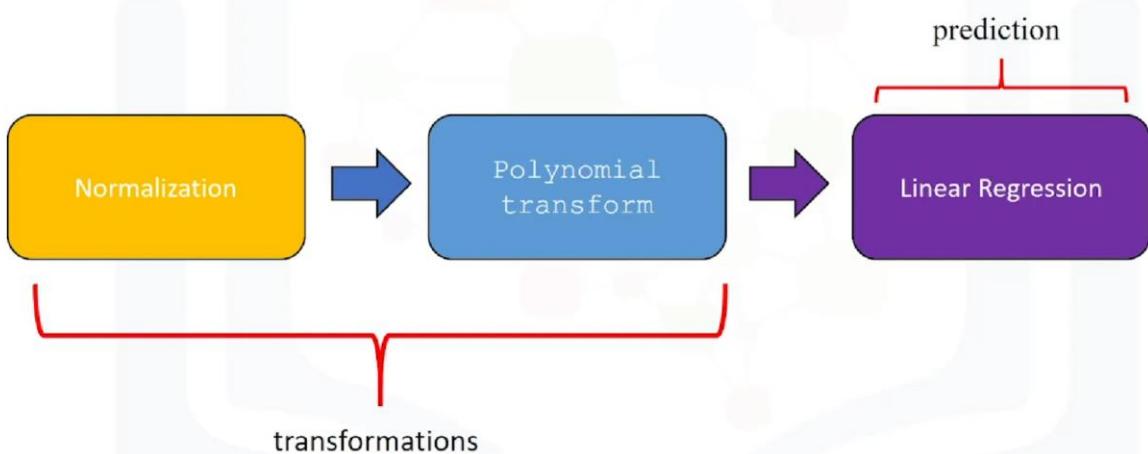
Pipelines

```
from sklearn.preprocessing import PolynomialFeatures  
from sklearn.linear_model import LinearRegression  
from sklearn.preprocessing import StandardScaler  
from sklearn.pipeline import Pipeline
```

Мы упрощаем этот процесс с помощью конвейера. Конвейер последовательно выполняет ряд преобразований. На последнем этапе выполняется предсказание.

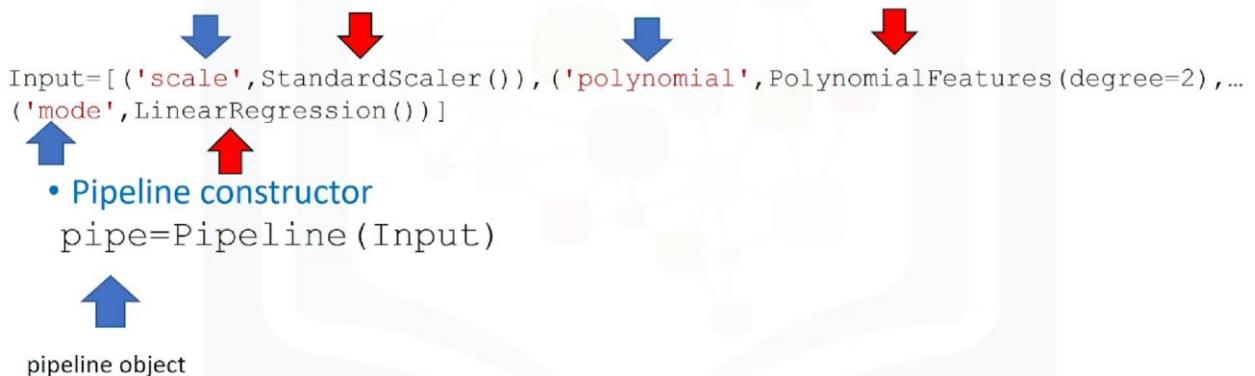
Pipelines

- There are many steps to getting a prediction



Сначала мы импортируем все необходимые нам модули, затем импортируем библиотечный конвейер.

Pipeline Constructor



Мы создаем список кортежей, первый элемент в кортеже содержит имя модуля оценщика. Второй элемент содержит конструктор модели.

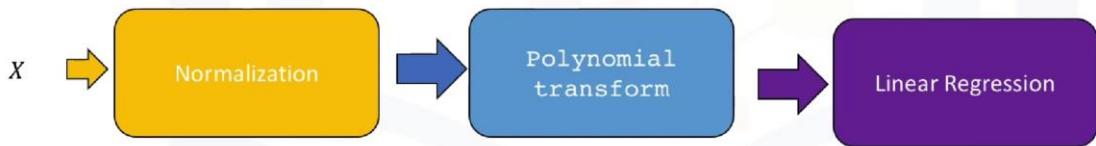
Мы вводим список в конструктор конвейера. Теперь у нас есть объект конвейера.

Мы можем обучить трубопровод, применив метод train к объекту трубопровода.

Pipeline Constructor

- We can train the pipeline object

```
Pipe.fit(df[['horsepower', 'curb-weight', 'engine-size', 'highway-mpg']],y)  
yhat=Pipe.predict(X[['horsepower', 'curb-weight', 'engine-size', 'highway-mpg']])
```

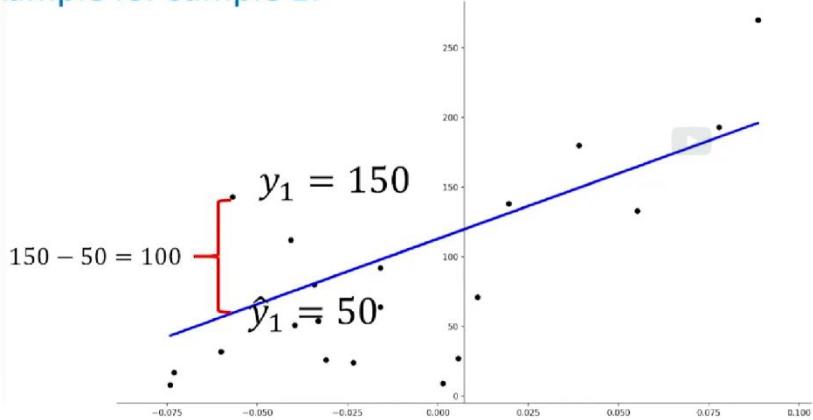


Мы также можем произвести предсказание. Метод нормализует данные, выполняет полиномиальное преобразование, затем выдает предсказание.

Теперь, когда мы увидели, как можно оценить модель с помощью визуализации, мы хотим численно оценить наши модели.

Mean Squared Error (MSE)

- For Example for sample 1:

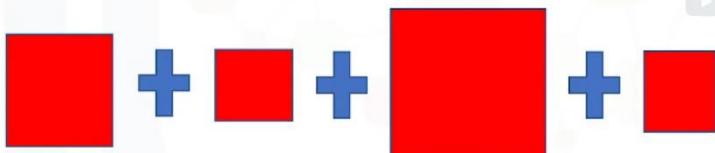


Давайте рассмотрим некоторые меры, которые мы используем для оценки на выборке.

Эти меры - способ численно определить, насколько хорошо модель подходит к нашим данным. **Два важных показателя, которые мы часто используем для**

определения соответствия модели, это: средняя квадратическая ошибка (MSE) и R-квадрат.

Mean Squared Error (MSE)



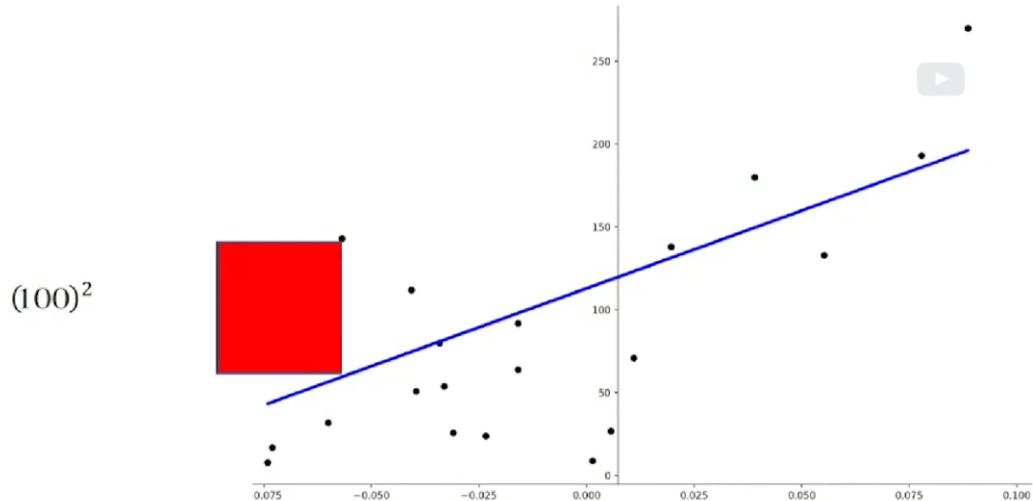
Number of Samples

Чтобы измерить MSE, мы находим разницу между фактическим значением y и предсказанным и прогнозируемым значением y , а затем возводим ее в квадрат.

В данном случае фактическое значение равно 150, а предсказанное - 50. Вычитая эти точки получаем 100.

Mean Squared Error (MSE)

- To make all the values positive we square it



Затем возводим это число в квадрат. Затем мы берем среднее значение или среднее всех ошибок, складывая их вместе и деля на количество образцов.

Чтобы найти MSE в Python, мы можем импортировать "mean_squared_error" из "scikit-learn.metrics".

Mean Squared Error (MSE)

- In python we can measure the MSE as follows

```
from sklearn.metrics import mean_squared_error  
  
mean_squared_error(df['price'], Y_predict_simple_fit)  
  
3163502.944639888
```

Функция "mean_Squared_error" получает два входа: фактическое значение целевой переменной и прогнозируемое значение целевой переменной.

R-квадрат также называется коэффициентом детерминации. Это мера, определяющая насколько близки данные к подогнанной линии регрессии. Итак, насколько близки наши фактические данные к нашей расчетной модели?

R-squared/ R²

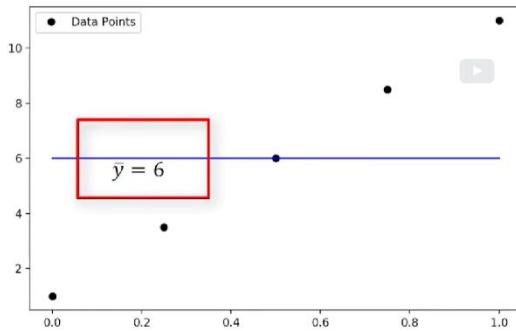
- The Coefficient of Determination or R squared (R²)
- Is a measure to determine how close the data is to the fitted regression line.
- R²: the percentage of variation of the target variable (Y) that is explained by the linear model.
- Think about as comparing a regression model to a simple model i.e the mean of the data points

Подумайте об этом как о сравнении регрессионной модели с простой моделью, т.е. средним значением точек данных. точки. Если переменная x является хорошим

предиктором, наша модель должна работать намного лучше, чем только среднее значение.

Coefficient of Determination (R^2)

- In this example the average of the data points \bar{y} is 6



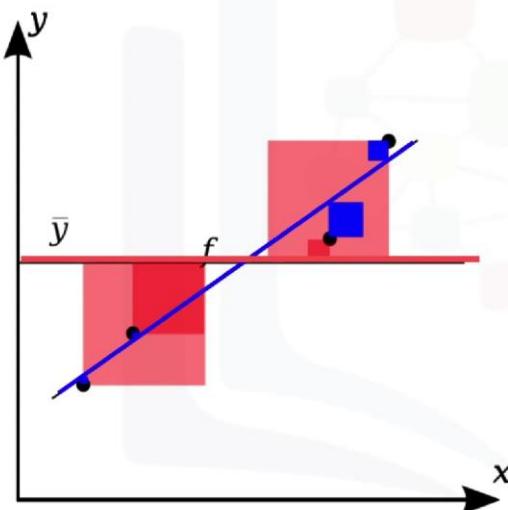
В данном примере среднее значение точек данных $y \mid 6$. Коэффициент детерминации R^2 - это 1 минус отношение MSE регрессии деленное на MSE среднего значения точек данных. В большинстве случаев он принимает значения между 0 и 1.

Coefficient of Determination (R^2)

$$R^2 = \left(1 - \frac{\text{MSE of regression line}}{\text{MSE of the average of the data}} \right)$$

Давайте рассмотрим случай, когда линия обеспечивает относительно хорошую подгонку.

Coefficient of Determination (R^2)



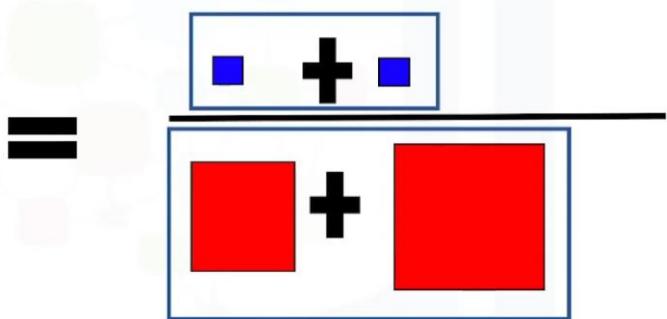
- The blue line represents the regression line
- The blue squares represents the MSE of the regression line
- The red line represents the average value of the data points
- The red squares represent the MSE of the red line
- We see the area of the blue squares is much smaller than the area of the red squares

- Синяя линия представляет собой линию регрессии.
- Синие квадраты представляют MSE линии регрессии.
- Красная линия представляет собой среднее значение точек данных.
- Красные квадраты представляют MSE красной линии.

Мы видим, что площадь синих квадратов намного меньше площади красных квадратов. В данном случае, поскольку линия хорошо подходит, средняя квадратичная ошибка мала, поэтому числитель мал.

Coefficient of Determination (R^2)

$$\frac{\text{MSE of regression line}}{\text{MSE of } \bar{y}} = 0$$

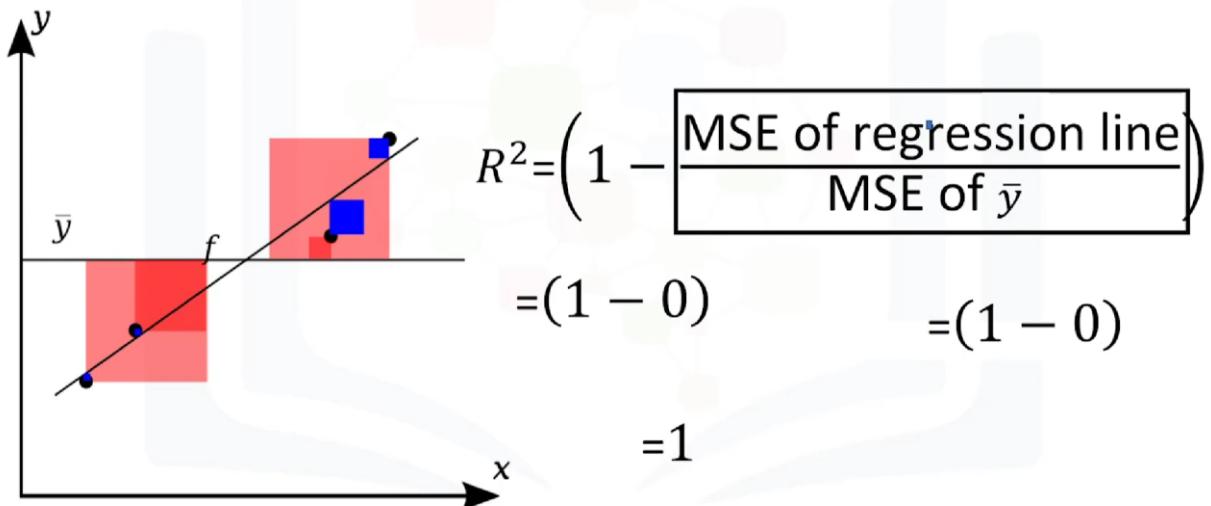


- Средняя квадратичная ошибка линии относительно велика, поэтому числитель большой.
- Маленькое число, деленное на большее число, становится еще меньше. Доведенное до крайности это значение стремится к нулю.

Если мы подставим значение R^2 из предыдущего слайда, то получим значение, близкое к единице это означает, что линия хорошо подходит к данным. Вот пример линии, которая плохо хорошо подходит к данным.

Если мы просто рассмотрим площадь красных квадратов по сравнению с синими, мы увидим, что площадь почти одинаковая.

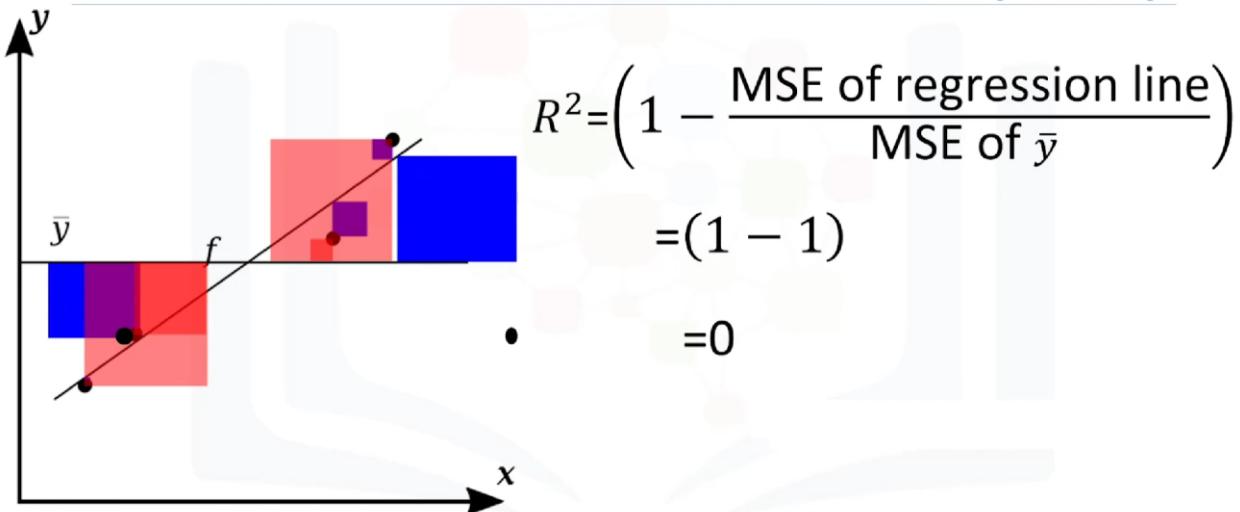
Coefficient of Determination (R^2)



Отношение площадей близко к единице. В этом случае R^2 близок к нулю.

Эта линия работает примерно так же, как и при использовании среднего значения точек данных, следовательно, эта линия не показала хороших результатов.

Coefficient of Determination (R^2)



Мы находим значение R -квадрат в Python с помощью метода оценки в объекте линейной регрессии объекта.

R-squared/ R²

- Generally the values of the MSE are between 0 and 1.
- We can calculate the R² as follows

```
X = df[['highway-mpg']]
```

```
Y = df['price']
```

```
lm.fit(X, Y)
```

```
lm.score(X, y)  
0.496591188
```

По значению, которое мы получаем из этого примера, можно сказать, что примерно 49,695% вариации цены объясняется этой простой линейной моделью.

Значение R² обычно находится в диапазоне от 0 до 1. Если R² отрицательное, это может быть связано с чрезмерной подгонкой модели, которую мы обсудим в следующем модуле.

ПРИНЯТИЕ РЕШЕНИЙ

Как мы можем определить, верна ли наша модель?

Первое, что вы должны сделать, это убедиться, что результаты вашей модели имеют смысл. Вы всегда должны использовать визуализацию, численные меры для оценки и сравнение между различными моделями.

Давайте рассмотрим пример предсказания. Если вы помните, мы обучали модель с помощью метода fit метод. Теперь мы хотим выяснить, какова будет цена автомобиля, у которого пробег на галлон по шоссе составляет 30 миль галлон 30. Подставив это значение в метод прогнозирования, мы получим цену в размере

\$13 771,30, что кажется вполне логичным. Например, значение не является отрицательным, чрезвычайно высоким или чрезвычайно низким. Мы можем посмотреть на коэффициенты, изучив атрибут подчеркивания coef_. Если вы вспомнить выражение для простой линейной модели, которая предсказывает цену

по количеству миль на галлон на шоссе.

Do the predicted values make sense

- First we train the model

```
lm.fit(df['highway-mpg'], df['prices'])
```

- Let's predict the price of a car with 30 highway-mpg.

```
lm.predict(np.array(30.0).reshape(-1,1))
```

- Result: \$ 13771.30

```
lm.coef_
-821.73337832
```

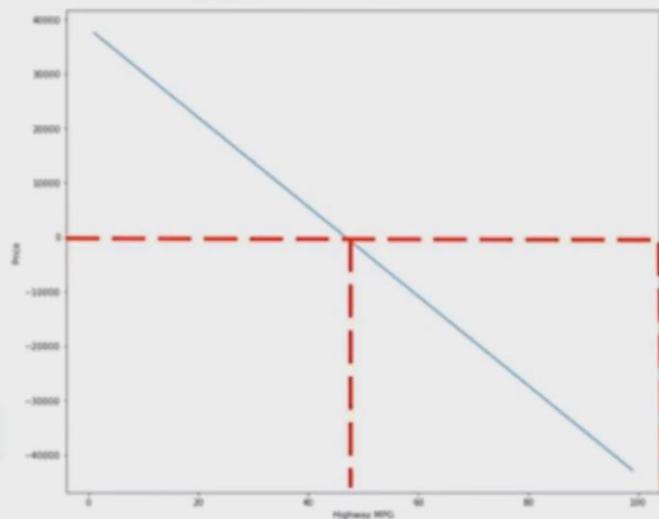
- Price = 38423.31 - 821.73 * highway-mpg

Это значение соответствует кратности признака high miles per gallon.

Таким образом, при увеличении на одну единицу количества миль по шоссе на галлон стоимость автомобиля уменьшается примерно на 821 доллар. Это значение также кажется разумным. Иногда ваша модель будет выдавать значения, которые не имеют смысла.

Например, если мы построим график модели для количества миль на галлон по шоссе в диапазоне от 0 до 100, мы получим отрицательные значения для цены.

Do the predicted values make sense



Это может быть связано с тем, что значения в этом диапазоне не реалистичны линейное предположение неверно или у нас нет данных по автомобилям в этом диапазоне. В этом случае маловероятно, что автомобиль будет иметь пробег топлива в этом диапазоне, поэтому наша модель кажется верной. Чтобы сгенерировать последовательность значений в указанном диапазоне

импортируйте numpy, затем используйте функцию numpy arrange для генерации последовательности последовательность начинается с 1 и увеличивается на 1, пока мы не достигнем 100.

Do the predicted values make sense

- First we import numpy

```
import numpy as np
```

- We use the numpy function arrange to generate a sequence from 1 to 100

```
new_input=np.arange(1,101,1).reshape(-1,1)
```



Первый параметр - это начальная точка последовательности. Второй параметр - конечная точка плюс один из последовательности. Последний параметр - размер шага между элементами последовательности.

В данном случае это один, поэтому мы увеличиваем последовательность на один шаг за раз от одного до двух и так далее.

Do the predicted values make sense

- We can predict new values

```
yhat=lm.predict(new_input)
```

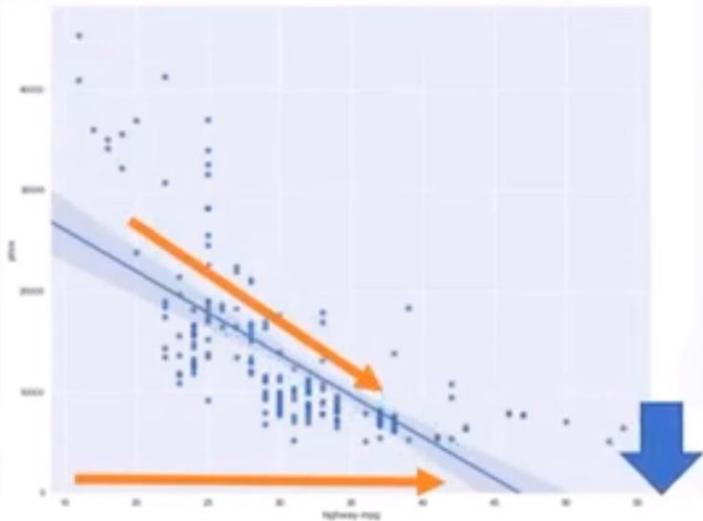
```
array([-37601.57247994, 36779.83910151, 35998.10572319, 35136.37234487,
       34314.63894655, 33492.90558823, 32671.1722099, 31849.43883158,
       31027.70545326, 30205.97207494, 29384.23869662, 28562.50531829,
       27746.77193997, 26919.03856165, 26097.30518333, 25275.57180501,
       24453.83842668, 23632.10504836, 22810.37167004, 21988.63829172,
       21166.9649134, 20345.17153508, 19523.43815679, 18701.70477943,
       17879.97140011, 17058.23802179, 16236.50464347, 15414.77126514,
       14593.03798682, 13771.3045085, 12949.57113018, 12127.83775186,
       11306.10437353, 10484.37099521, 9662.63761689, 8840.90423857,
       8019.17086025, 7197.43748192, 6375.7041036, 5553.97072528,
       4732.23734696, 3910.50396864, 3088.77059031, 2267.03721199,
       1445.30383367, 623.57045533, -158.16292297, -1019.8963013,
       -1841.62987962, -2663.36305794, -3485.09643626, -4356.82981458,
       -5128.5631928, -5950.29657123, -6772.02994895, -7593.76332787,
       -8415.49670619, -9237.23008451, -10059.96346289, -10880.69484116,
       -11792.43021948, -12524.14359978, -13345.89997812, -14167.63039445,
       -14989.36373277, -15811.0936109, -16622.83048941, -17454.56386773,
       -18276.3935811, -19088.2006214, -19901.0077517, -20741.49738102,
       -21563.23075934, -22384.96413767, -23206.49751599, -24028.03042332,
       -24851.16427263, -25671.89165098, -26493.63102997, -27315.3644077,
       -28137.09778592, -28958.83116423, -29780.56454258, -30602.29792088,
       -31424.03129921, -32245.76446793, -33067.49805585, -33889.23143417,
       -34710.96481248, -35532.69819098, -36354.43156924, -37176.16494746,
       -37997.89832578, -38819.6317041, -39641.36509243, -40463.09846075,
       -41284.83183907, -42106.56523739, -42928.29899571])
```

Мы можем использовать выход для предсказания новых значений, выход представляет собой массив numpy, многие из значений являются отрицательные. Использование графика регрессии для визуализации данных - это первый метод, который вы должны попробовать.

Примеры построения графика полиномиальной регрессиисмотрите в лабораторных работах.

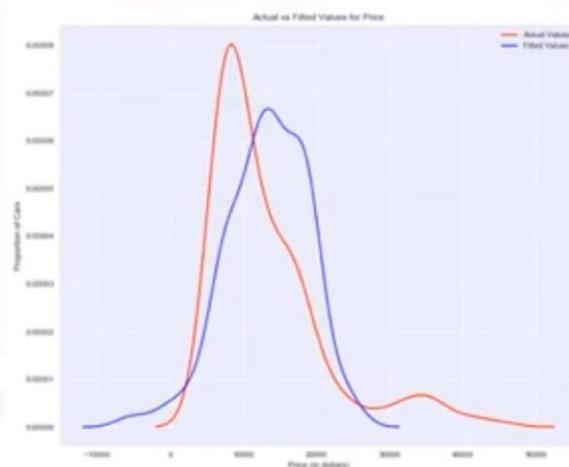
Visualization

- Simply visualizing your data with a regression



В данном примере влияние независимой переменной очевидно. Данные имеют тенденцию к снижению по мере увеличения зависимой переменной, график также демонстрирует нелинейное поведение.

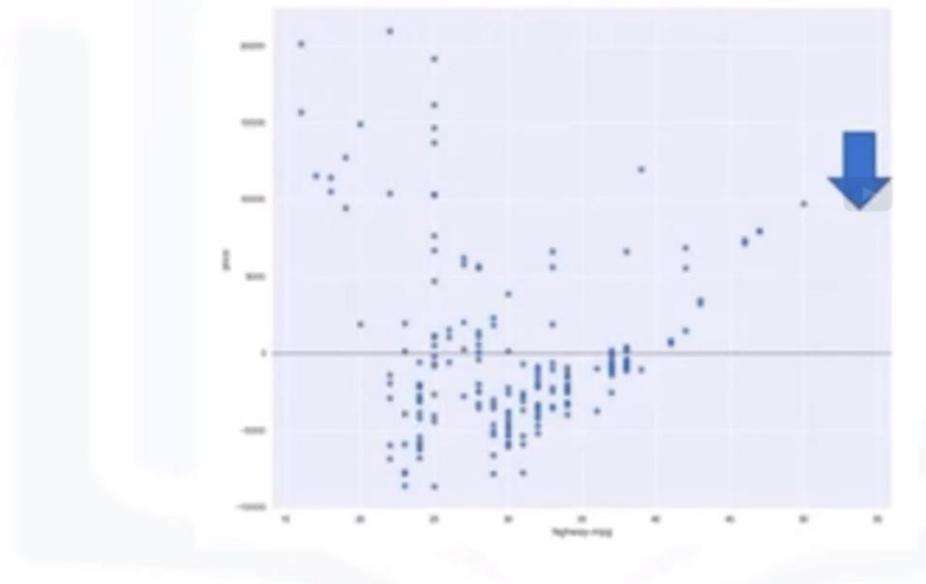
Visualization



Изучая график остатков, мы видим, что в данном случае остатки имеют кривизну, указывающую на нелинейное поведение. График распределения - хороший метод для множественной линейной регрессии.

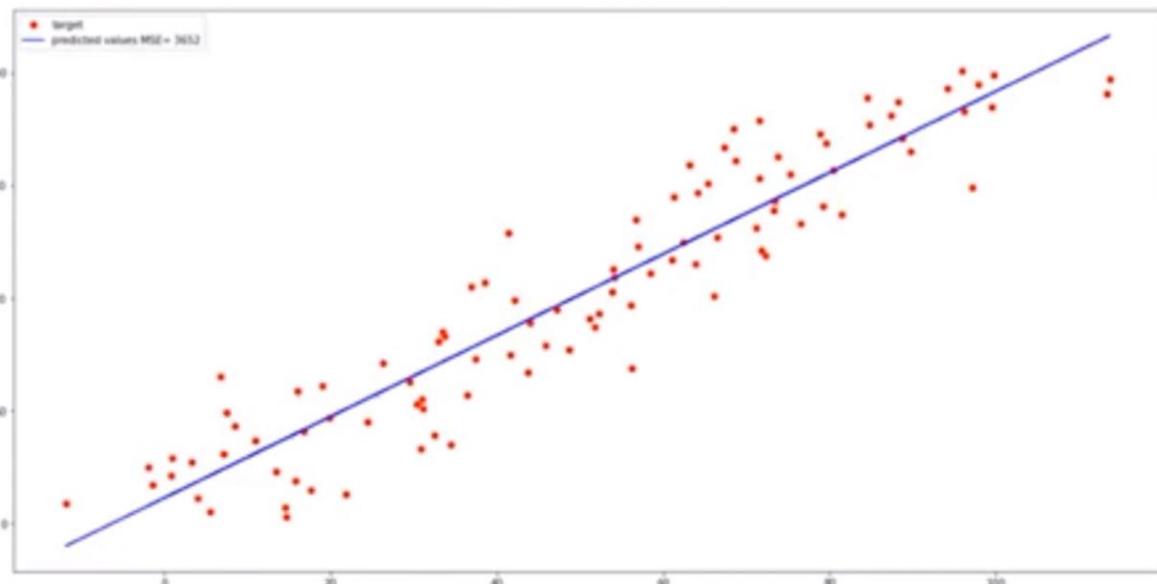
Например, мы видим, что предсказанные значения для цен в диапазоне от тридцати тысяч до пятидесяти тысяч неточны, это говорит о том, что нелинейная модель может быть более подходящей или нам нужно больше данных в этом диапазоне.

Residual Plot



Среднеквадратичная ошибка, возможно, является наиболее интуитивно понятной численной мерой для определения того хороша ли модель или нет. Давайте посмотрим, как различные показатели среднеквадратичной ошибки влияют на модель.

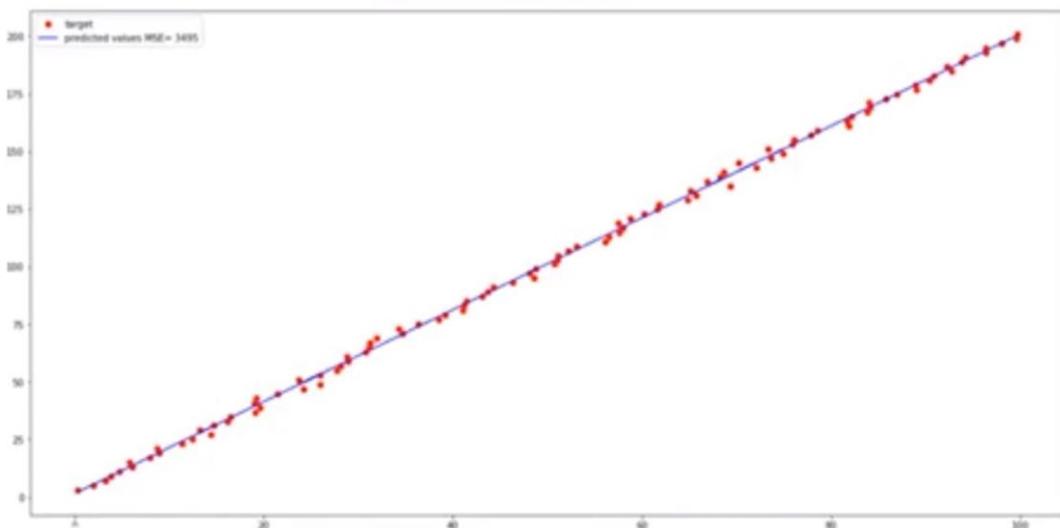
На рисунке показан пример со средней квадратической ошибкой 3495.



Этот пример имеет среднюю квадратичную ошибку составляет три тысячи шестьсот пятьдесят два. На итоговом графике средняя квадратическая ошибка составляет двенадцать тысяч восемьсот семьдесят. По мере увеличения квадратичной ошибки цели удаляются от от прогнозируемых точек. Как мы уже говорили, квадрат r^2 это еще один популярный метод оценки модели.

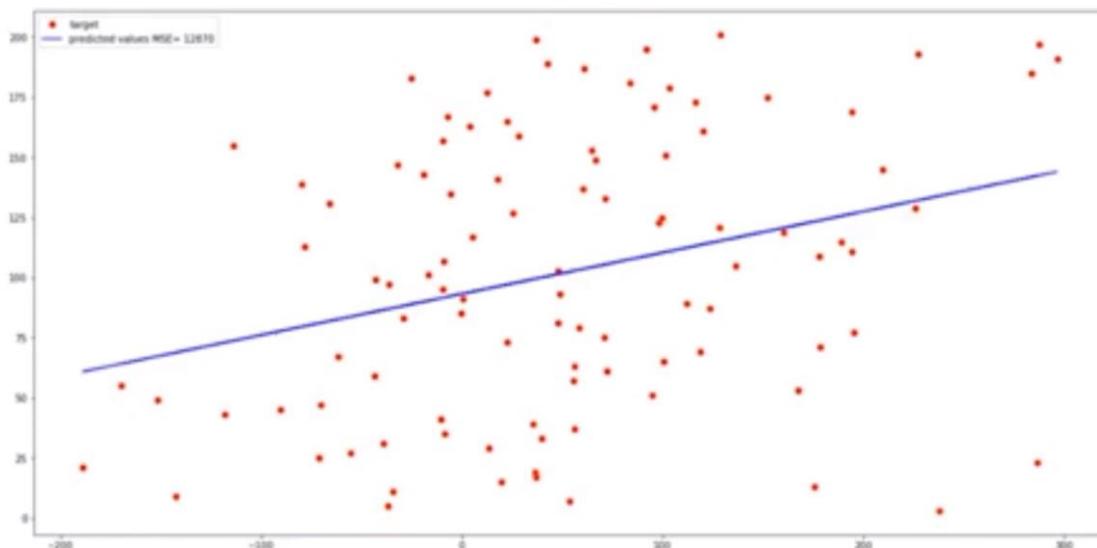
Он показывает, насколько хорошо ваша линия вписывается в модель. Значения квадрата r **варьируются от нуля до единицы. R квадрат** говорит нам о том, какой процент изменчивости зависимой переменной учитывает регрессией на независимую переменную.

Numerical measures for Evaluation



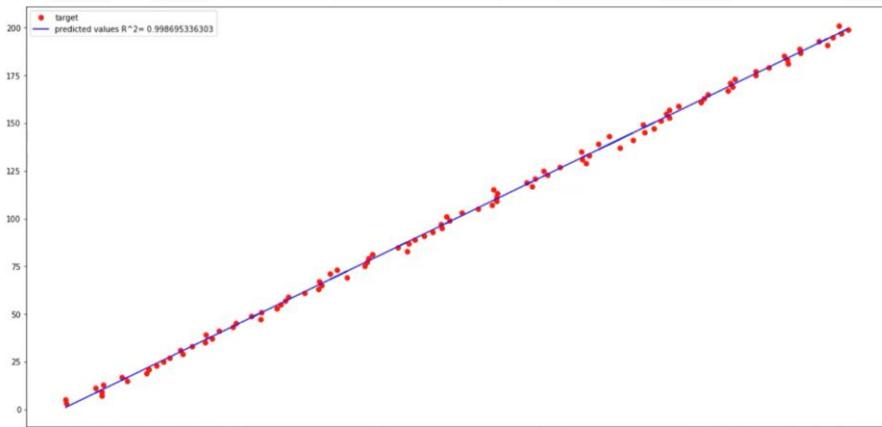
Квадрат r , равный 1, означает, что все изменения другой зависимой переменной полностью объясняются изменениями независимых переменных.

Numerical measures for Evaluation



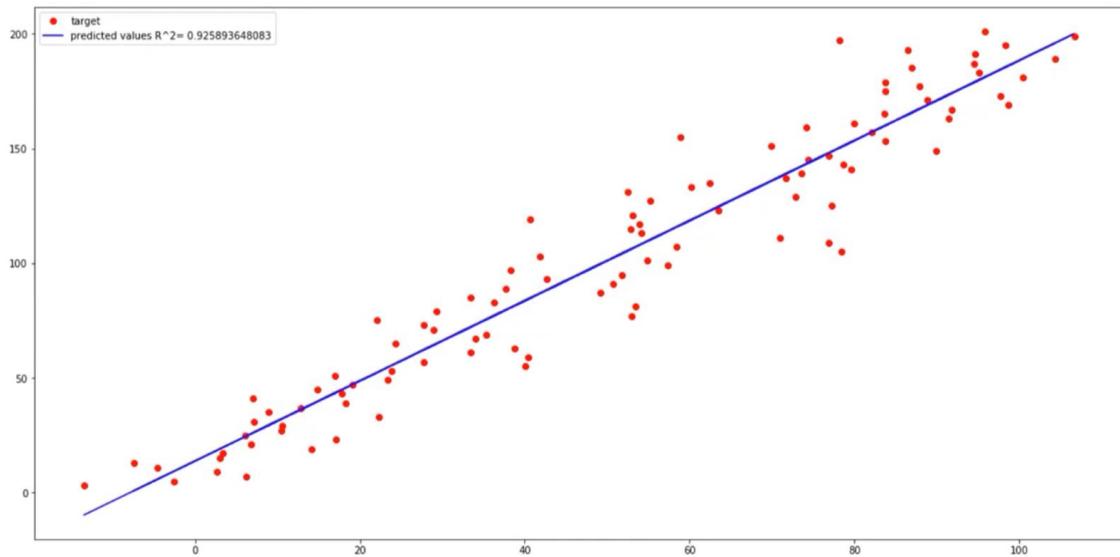
На этом графике целевые точки выделены красным цветом, а прогнозируемая линия - синим. При r квадрат 0,9986 модель хорошо подходит.

Numerical measures for Evaluation



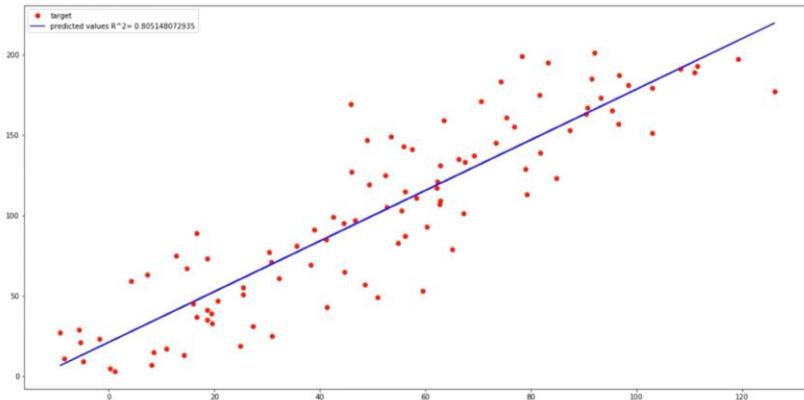
Это означает, что более 99 изменчивости прогнозируемой переменной объясняется независимыми переменными. Эта модель имеет r квадрат 0,9226, то есть по-прежнему существует сильная линейная связь, и модель по-прежнему хорошо подходит.

r squared 0806 из данных мы можем визуально видеть, что значения разбросаны вокруг линии.



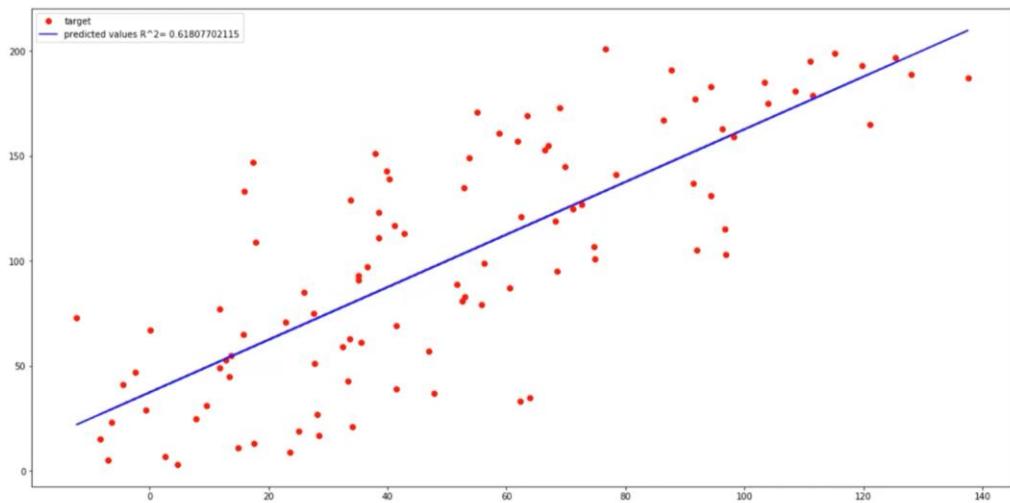
Они все еще близки к линии, и мы можем сказать, что 80 процентов изменчивости прогнозируемой переменной объясняется независимыми переменными.

Numerical measures for Evaluation



А квадрат r 0,61 означает, что приблизительно 61 процент наблюдаемой вариации может быть объяснен независимыми переменными.

Numerical measures for Evaluation



Приемлемое значение квадрата r зависит от того, какую область вы изучаете и каков ваш случай использования что вы изучаете и какова ваша цель. Falcon Miller 1992 считает, что приемлемое значение r квадрат должно быть не менее 0,1.

Означает ли меньшая средняя квадратическая ошибка лучшую подгонку? Не обязательно MSE4 и MLR модели будет меньше чем MSE для модели SLR. Поскольку ошибки данных будут уменьшаться, когда больше переменных, включенных в модель. Полиномиальная регрессия также будет иметь меньший MSE, чем обычная регрессия.

В этом уроке вы узнали, как:

Определять объясняющую переменную и переменную отклика:

- Определять переменную отклика (y) как объект эксперимента и объясняющую переменную (x) как переменную, используемую для объяснения изменения переменной отклика. Поймите разницу между простой линейной регрессией, поскольку она касается изучения только одной объясняющей переменной, и множественной линейной регрессией, поскольку она касается изучения двух или более объясняющих переменных.
- Оценивать модель с помощью визуализации: Визуально представляя ошибки переменной с помощью диаграмм рассеяния и интерпретируя результаты модели.
- Определять альтернативные подходы к регрессии: Использовать полиномиальную регрессию, когда линейная регрессия не отражает криволинейную зависимость между переменными, и как выбрать оптимальный порядок для использования в модели.
- Интерпретировать R-квадрат и среднюю квадратичную ошибку: Интерпретируйте R-квадрат ($\times 100$) как процент вариации переменной ответа y , которая объясняется вариацией объясняющей переменной (переменных) x . Средняя квадратичная ошибка показывает, насколько близко линия регрессии подходит к набору точек. Для этого берутся средние расстояния от фактических точек до прогнозируемых точек и возводятся в квадрат.

ОЦЕНКА И УТОЧНЕНИЕ МОДЕЛИ

Оценка модели говорит нам о том, как работает наша модель в реальном мире.

В предыдущем модуле мы говорили об оценке на выборке.

Оценка на выборке говорит нам о том, насколько хорошо наша модель соответствует данным, которые уже были предоставлены для ее обучения.

Она не дает нам оценки того, насколько хорошо обучаемая модель может предсказывать новые данные.

Решение состоит в том, чтобы разделить наши данные, использовать данные из выборки или тренировочные данные для обучения модели.

Model Evaluation

- In-sample evaluation tells us how well our model will fit the data used to train it
- Problem?
 - It does not tell us how well the trained model can be used to predict new data
- Solution?
 - In-sample data or training data
 - Out-of-sample evaluation or test set

Остальные данные, называемые тестовыми, используется как данные вне выборки. Эти данные затем используются для приблизительной оценки того, как модель работает в реальном мире.

Разделение данных на обучающие и тестовые наборы является важной частью оценки модели. Мы используем тестовые данные, чтобы получить представление о том, как наша модель будет работать в реальном мире. **Когда мы разделяем набор данных, обычно большая часть данных используется для обучения, а меньшая часть используется для тестирования.**

Например, мы можем использовать 70 процентов данных для обучения.

Training/Testing Sets

Data:

- Split dataset into:
 - Training set (70%), 
 - Testing set (30%) 
- Build and train the model with a training set
- Use testing set to assess the performance of a predictive model
- When we have completed testing our model we should use all the data to train the model to get the best performance

Затем мы используем 30 процентов для тестирования.

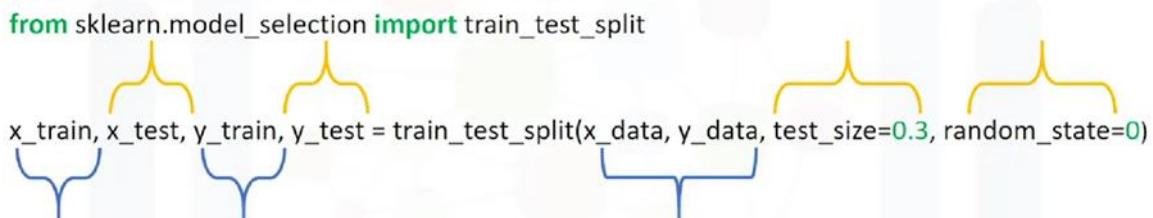
- Мы используем обучающее множество для построения модели и выявления прогнозируемых взаимосвязей.
- Затем мы используем набор для тестирования, чтобы оценить эффективность модели.
- Когда мы закончим тестирование нашей модели, мы должны использовать все данные для обучения модели.

Популярной функцией в пакете scikit-learn для разбиения наборов данных является функция train test split.

Function train_test_split()

- Split data into random train and test subsets

```
from sklearn.model_selection import train_test_split  
x_train, x_test, y_train, y_test = train_test_split(x_data, y_data, test_size=0.3, random_state=0)
```



- **x_data:** features or independent variables
- **y_data:** dataset target: df['price']
- **x_train, y_train:** parts of available data as training set
- **x_test, y_test:** parts of available data as testing set
- **test_size:** percentage of the data for testing (here 30%)
- **random_state:** number generator used for random sampling

Эта функция случайным образом разбивает набор данных на обучающее и тестирующее подмножества. Из примера фрагмента кода, этот метод импортирован из sklearn.cross-validation.

- Входные параметры y_data - это целевая переменная. В примере с оценкой автомобиля это будет цена,
- а x_data, список прогнозируемых переменных. В данном случае это все остальные переменные в наборе данных по автомобилям, которые мы используем для прогнозирования цены.

На выходе получается массив.

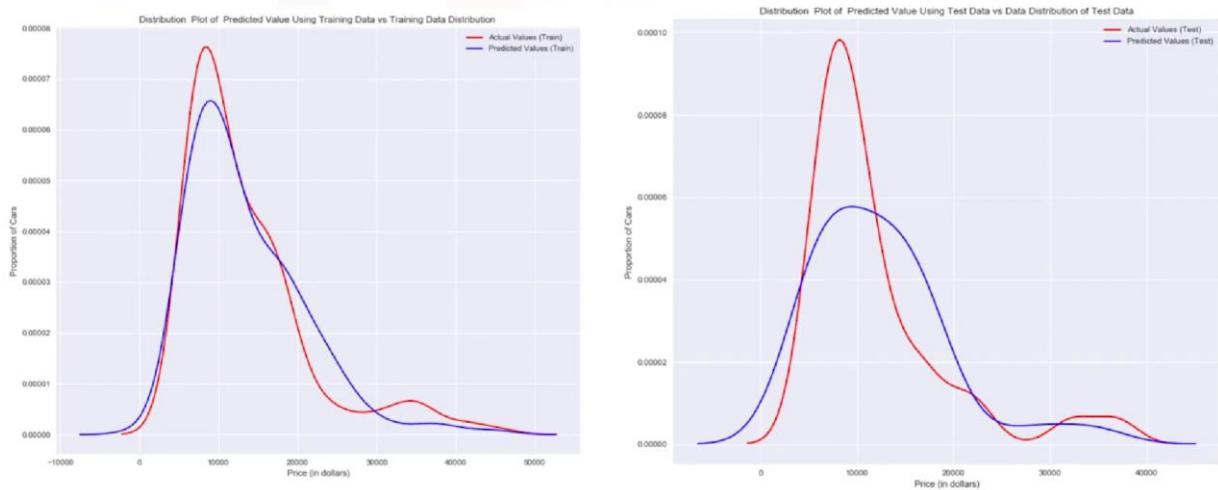
- x_train и y_train - подмножества для обучения.
- x_test и y_test - подмножества для тестирования.
- **В данном случае размер теста (test_size = 0.3) - это процент от данных для набора тестирования. Здесь он составляет 30 (0.3) процентов.**

- **Случайное состояние** (`random_state`) - это случайная затравка для случайного разбиения набора данных.

Ошибка обобщения - это мера того, насколько хорошо наши данные предсказывают ранее невидимые данные.

Ошибка, которую мы получаем, используя наши тестовые данные, является аппроксимацией этой ошибки.

Generalization Error



На этом рисунке показано **распределение фактических значений красным цветом**, которое сравнивается с **предсказанными значениями из линейной регрессии - синий цвет**.

Мы видим, что распределения несколько похожи.

Если мы построим такой же график, используя тестовые данные (**график справа**), мы увидим, что распределения относительно разные. **Разница обусловлена ошибкой обобщения и представляет собой то, что мы видим в реальном мире.**

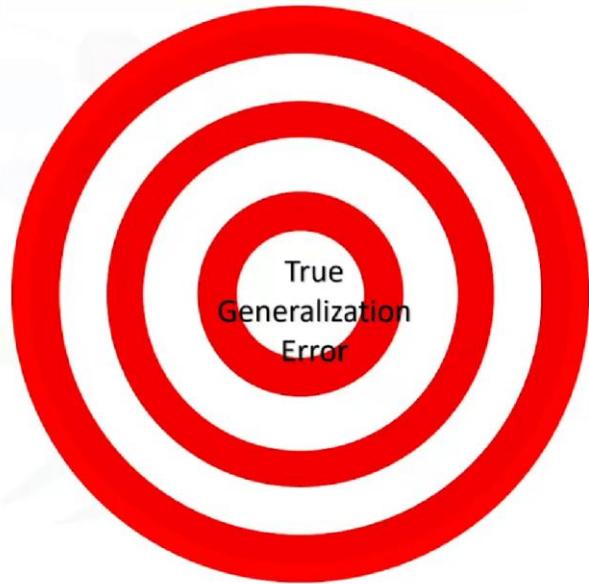
Использование большого количества данных для обучения дает нам точный способ определения того, насколько хорошо наша модель будет работать в реальном мире. Но точность результатов будет низкой.

Давайте поясним это на примере.

Lots of Training Data



Model



Центр этого "бычьего глаза" представляет собой правильную ошибку обобщения. Допустим, мы берем случайную выборку данных, используя 90 процентов данных для обучения и 10 процентов для тестирования.

В первый раз, когда мы экспериментируем, мы получаем хорошую оценку обучающих данных. Если мы повторим эксперимент, обучая модель с другой комбинацией выборок, мы также получим хороший результат.

Но результаты будут отличаться по сравнению с первым экспериментом. Повторяя эксперимент еще раз с другой комбинацией обучающих и тестовых выборок, результаты будут относительно близки к ошибке обобщения, но отличаются друг от друга.

Повторяя этот процесс, мы получаем хорошее приближение к ошибке обобщения, но точность низкая, т.е. все результаты сильно отличаются друг от друга.

Lots of Training Data

Model →



Если мы используем меньше точек данных для обучения модели и больше для ее тестирования, точность обобщения будет меньше, но модель будет иметь хорошую точность.

Lots of Training Data



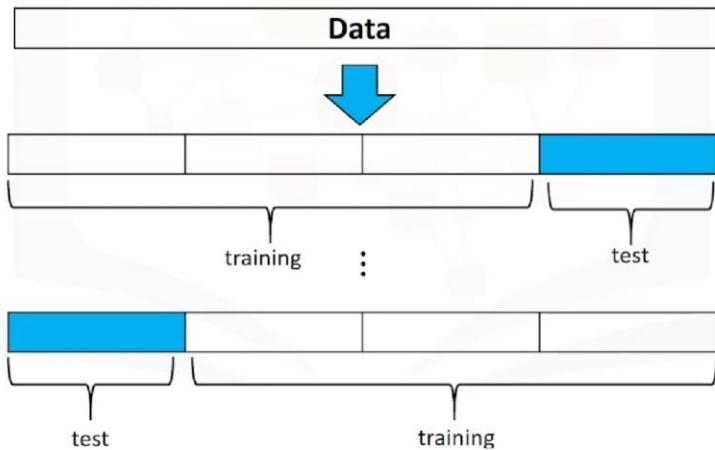
Рисунок выше демонстрирует это. Все наши оценки ошибок относительно близки друг к другу, но они находятся дальше от истинной эффективности обобщения.

Чтобы преодолеть эту проблему, мы используем перекрестную валидацию.

Одной из наиболее распространенных метрик оценки вне выборки является кросс-валидация. В этом методе набор данных разбивается на N равных групп.

Cross Validation

- Most common out-of-sample evaluation metrics
- More effective use of data (each observation is used for both training and testing)



Каждая группа называется складкой. Например, четыре складки.

- Часть складок может быть использована в качестве обучающего набора, который мы используем для обучения модели,
- а оставшиеся части - в качестве тестового набора, который мы используем для тестирования модели.

Например, мы можем использовать три складки для обучения, затем использовать одну складку для тестирования.

Это повторяется до тех пор, пока каждый раздел не будет использован как для обучения, так и для тестирования. В конце мы используем средние результаты в качестве оценки ошибки вне выборки.

Метрика оценки зависит от модели, например, r-квадрат.

Самый простой способ применения кросс-валидации - вызвать функцию **`cross_val_score`**, которая выполняет несколько оценок вне выборки.

Function `cross_val_score()`

```
from sklearn.model_selection import cross_val_score
```

```
scores= cross_val_score(lr, x_data, y_data, cv=3)
```

The diagram shows the execution flow of the `cross_val_score` function. It consists of two main parts: a downward-pointing yellow arrow above the function call, and two upward-pointing blue arrows below it. The first blue arrow points to the `cv=3` parameter in the function call, indicating that this value is passed to the function. The second blue arrow points to the `np.mean(scores)` line, indicating that this is the expression being evaluated after the function call.

Этот метод импортируется из пакета `sklearn` для выбора модели. Затем мы используем функцию `cross_val_score`, предварительно обозначив переменную, в которую мы положим результат, который вернет функция (в нашем случае это `score`)

Первые входные параметры (в скобках) – это тип модели, которую мы используем для кросс-валидации.

- В этом примере мы инициализируем модель линейной регрессии или объект `lr`, который мы передали функции `cross_val_score`.
- **`x_data`** - это данные прогнозируемой (независимой) переменной,
- **`y_data`**, данные целевой (зависимой) переменной (в нашем случае это цена).
- Мы можем управлять количеством разделов с помощью параметра `cv`.
Здесь `cv` равно трем, что означает, что набор данных разбит на три равных раздела.

Функция **возвращает массив оценок**, по одной для каждого раздела, который был выбран в качестве тестового набора.

Мы можем усреднить результат, чтобы оценить вне выборочного R-square с помощью функции среднего значения из библиотеки NumPy, в данном случае эта функция `np.mean(аргумент)`.

Алгоритм:

1. Сначала мы разбиваем данные на три складки. Две складки мы используем для обучения, оставшуюся складку - для тестирования.
2. Модель выдает результат. Мы будем использовать этот результат для расчета оценки. В случае R-square, т.е. коэффициент детерминации, мы сохраним это значение в массиве.
3. Мы повторим процесс, используя две складки для обучения и одну складку для тестирования.

4. Сохраним результат, затем используем другую комбинацию для обучения и оставшуюся складку для тестирования

Function cross_val_score()



5. Сохраняем окончательный результат. Функция **cross_val_score** возвращает значение оценки, чтобы сообщить нам результат кросс-валидации.

А что если нам нужно немного больше информации?

Что если мы хотим узнать фактические предсказанные значения полученных нашей моделью до того, как будут вычислены значения R-square?

Для этого мы используем функцию **cross_val_predict**. Входные параметры точно такие же, как и у функции **cross_val_score**, но на выходе получается предсказание.

Function cross_val_predict()

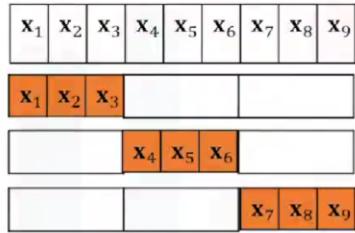
- It returns the prediction that was obtained for each element when it was in the test set
- Has a similar interface to cross_val_score()

```
from sklearn.model_selection import cross_val_predict
```

```
yhat= cross_val_predict (lr2e, x_data, y_data, cv=3)
```

Давайте проиллюстрируем процесс.

Function cross_val_predict()



Model

- Сначала мы разбиваем данные на три складки. Две складки мы используем для обучения, оставшуюся складку - для тестирования.
- Модель выдаст результат, и мы сохраним его в массиве. Мы повторим процесс, используя две складки для обучения и одну для тестирования.

Модель снова выдает результат. Наконец, мы используем последние две складки для обучения.

Затем мы используем данные для тестирования. Эта последняя тестовая складка выдает результат.

Function cross_val_predict()



Model

\hat{y}_4	\hat{y}_5	\hat{y}_6
-------------	-------------	-------------

\hat{y}_7	\hat{y}_8	\hat{y}_9
-------------	-------------	-------------

Function cross_val_predict()

x ₁	x ₂	x ₃	x ₄	x ₅	x ₆	x ₇	x ₈	x ₉
----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------

Model

\hat{y}_1	\hat{y}_2	\hat{y}_3
-------------	-------------	-------------

\hat{y}_4	\hat{y}_5	\hat{y}_6
-------------	-------------	-------------

\hat{y}_7	\hat{y}_8	\hat{y}_9
-------------	-------------	-------------

Эти прогнозы хранятся в массиве.

ПЕРЕОСНАЩЕНИЕ, НЕДООСНАЩЕНИЕ И ВЫБОР МОДЕЛИ

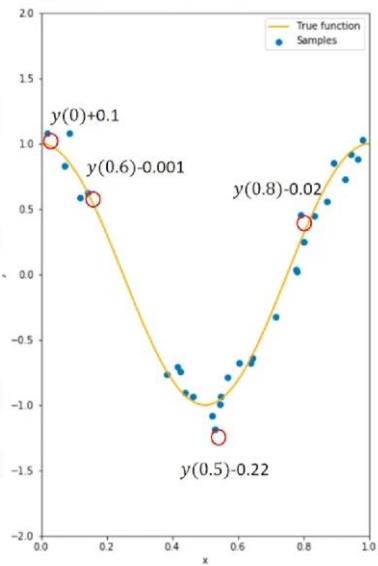
Если вы помните, в предыдущем модуле мы обсуждали полиномиальную регрессию. В этом разделе мы обсудим, как выбрать наилучший порядок полинома и проблемы, которые возникают при выборе полинома неправильного порядка.

Рассмотрим следующую функцию, мы предполагаем, что обучающие точки получены из полиномиальной функции плюс некоторый шум. Целью выбора модели является определение порядка полинома, чтобы обеспечить наилучшую оценку функции $y(x)$.

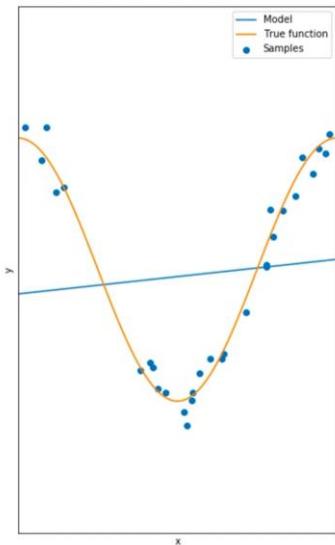
Если мы попытаемся подогнать функцию под линейную функцию, линия окажется недостаточно сложной, чтобы соответствовать данным.

Model Selection

$y(x) + \text{noise}$

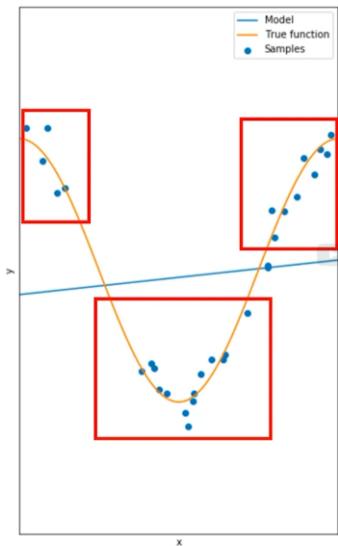


$y = b_0 + b_1 x$



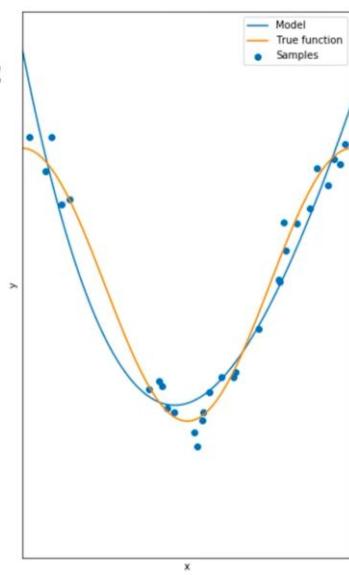
В результате возникает множество ошибок. Это называется недоподгонка, когда модель слишком проста, чтобы соответствовать данным.

$$y = b_0 + b_1 x$$

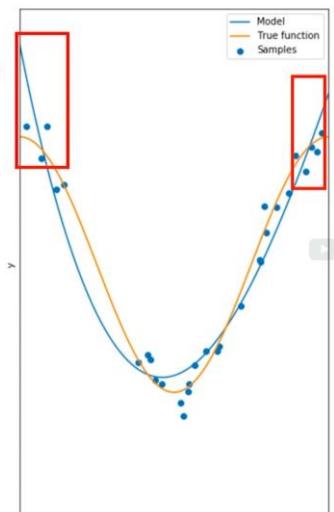


Если мы увеличим порядок полинома, модель подходит лучше, но модель все еще недостаточно гибкая и демонстрирует недостаточную подгонку.

$$y = b_0 + b_1 x + b_2 x^2$$

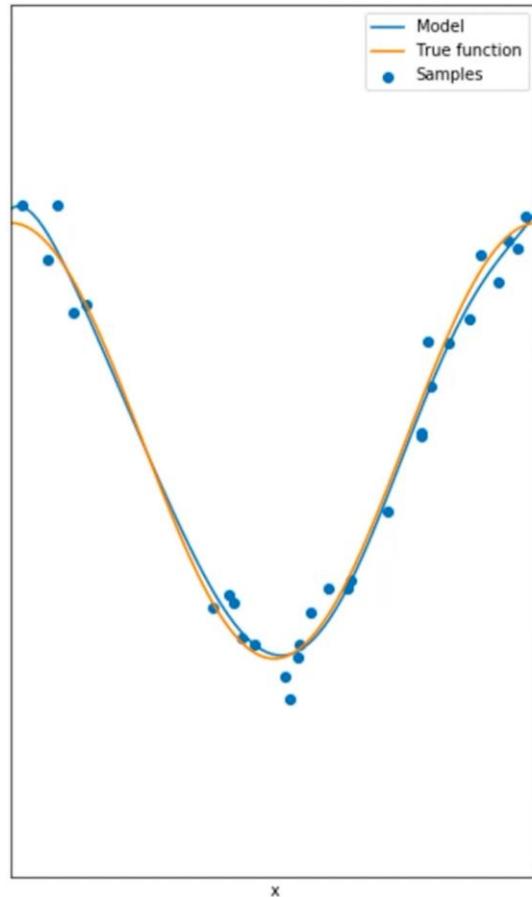


$$y = b_0 + b_1 x + b_2 x^2$$

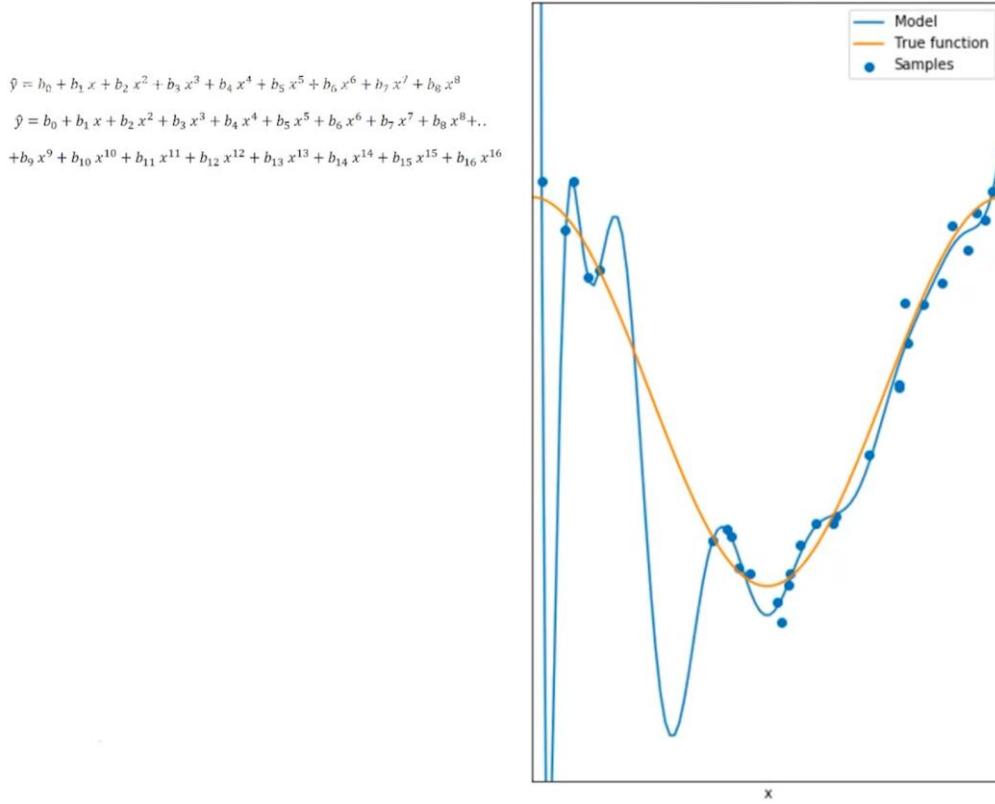


Это пример полинома 8-го порядка, используемого для подгонки данных.

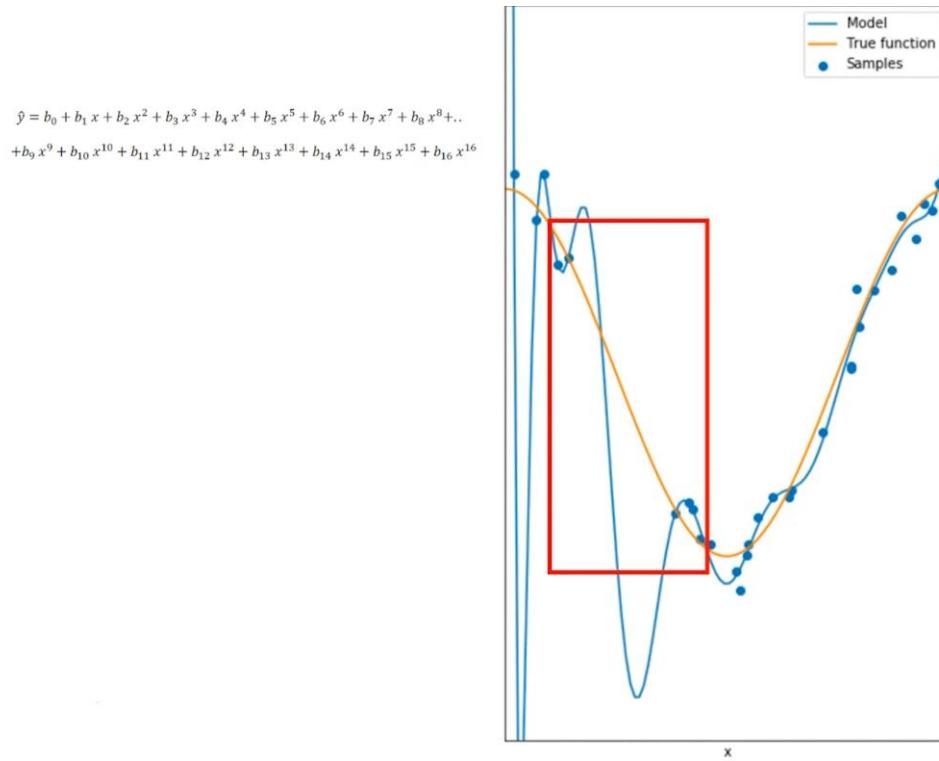
$$\hat{y} = b_0 + b_1 x + b_2 x^2 + b_3 x^3 + b_4 x^4 + b_5 x^5 + b_6 x^6 + b_7 x^7 + b_8 x^8$$



Мы видим, что модель хорошо подходит к данным и оценивает функции даже в точках перегиба. При увеличении полинома до полинома 16-го порядка, модель очень хорошо справляется с отслеживанием, но плохо справляется с оценкой функции. Это особенно заметно при малом количестве обучающих данных. Оцениваемая функция колеблется, не отслеживая функцию.

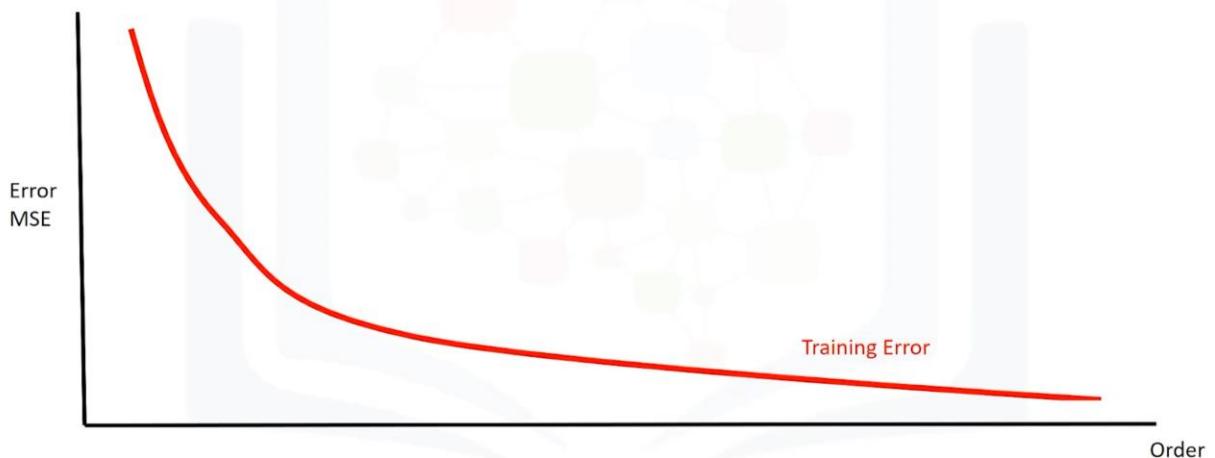


Это называется избыточной подгонкой, когда модель слишком гибкая и соответствует шуму, а не функции.



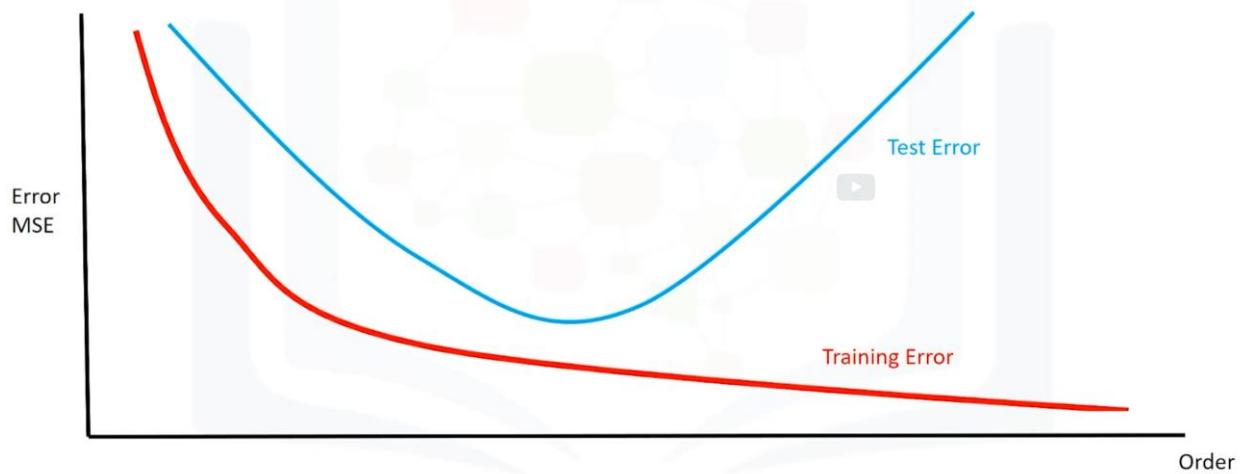
Давайте посмотрим на график среднеквадратичной ошибки для обучающего и тестового набора полиномов разного порядка. Горизонтальная ось представляет собой порядок полинома. Вертикальная ось - среднеквадратичная ошибка.

Model Selection



Ошибка обучения уменьшается с ростом порядка полинома. Ошибка тестирования является лучшим средством оценки ошибки полинома.

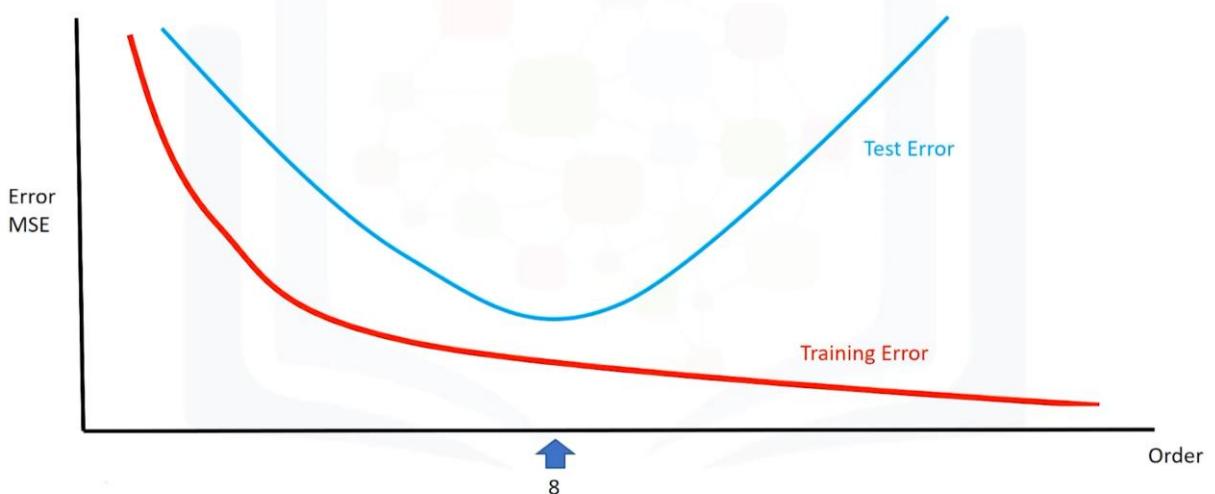
Model Selection



Ошибка уменьшается до тех пор, пока не будет определен наилучший порядок полинома. определяется. Затем ошибка начинает увеличиваться.

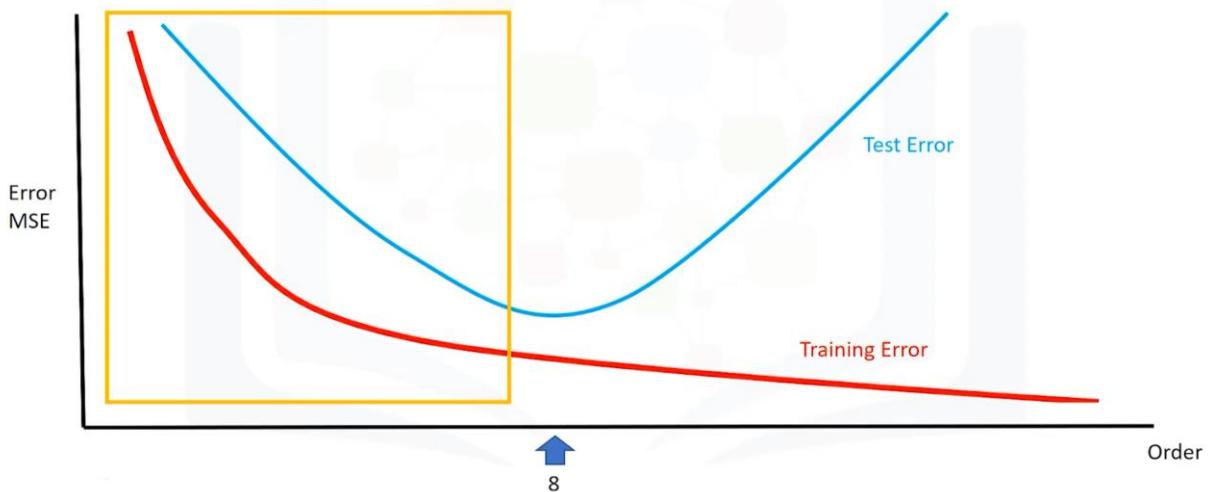
Мы выбираем порядок, который минимизирует ошибку теста. В данном случае это было 8. Все, что слева, считается недооценкой.

Model Selection



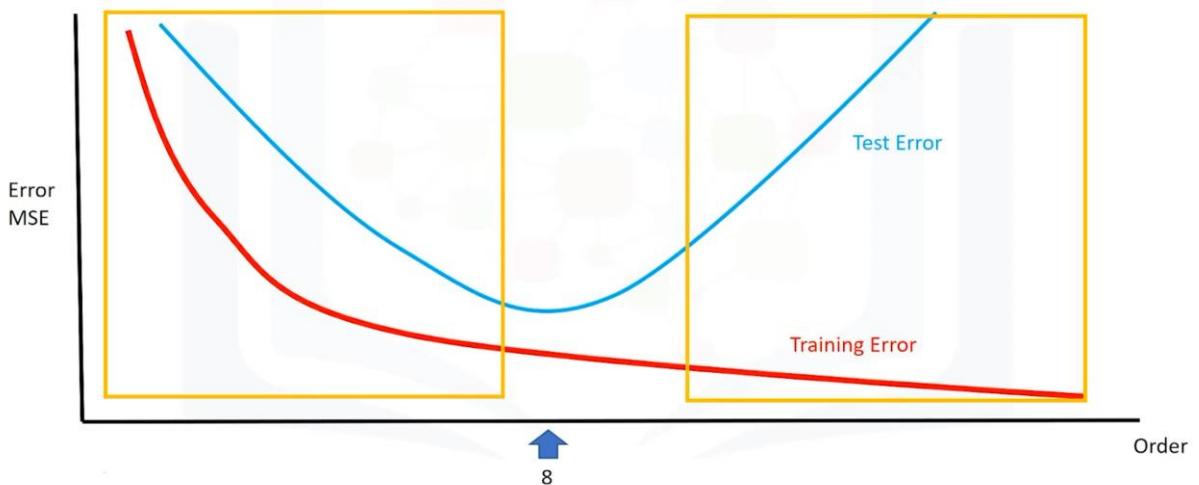
Все, что справа, - это перебор. Если мы выберем наилучший порядок полинома, мы все равно будем иметь некоторые ошибки. Если вспомнить исходное выражение для обучающих точек, то мы увидим шумовой член.

Model Selection



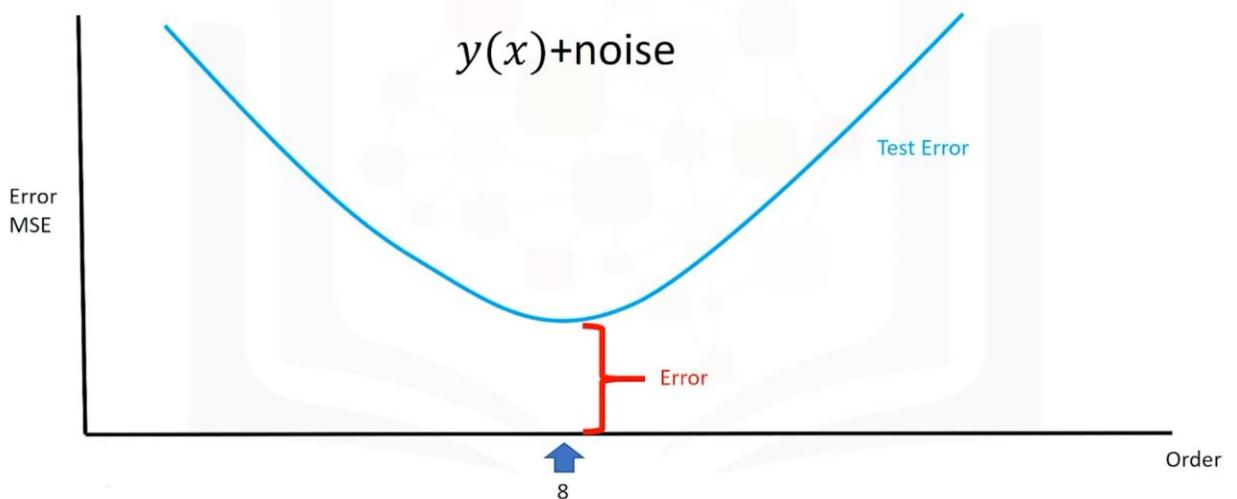
Этот член является одной из причин ошибки. Это происходит потому, что шум является случайным, и мы не можем его предсказать. Иногда это называют несводимой ошибкой.

Model Selection



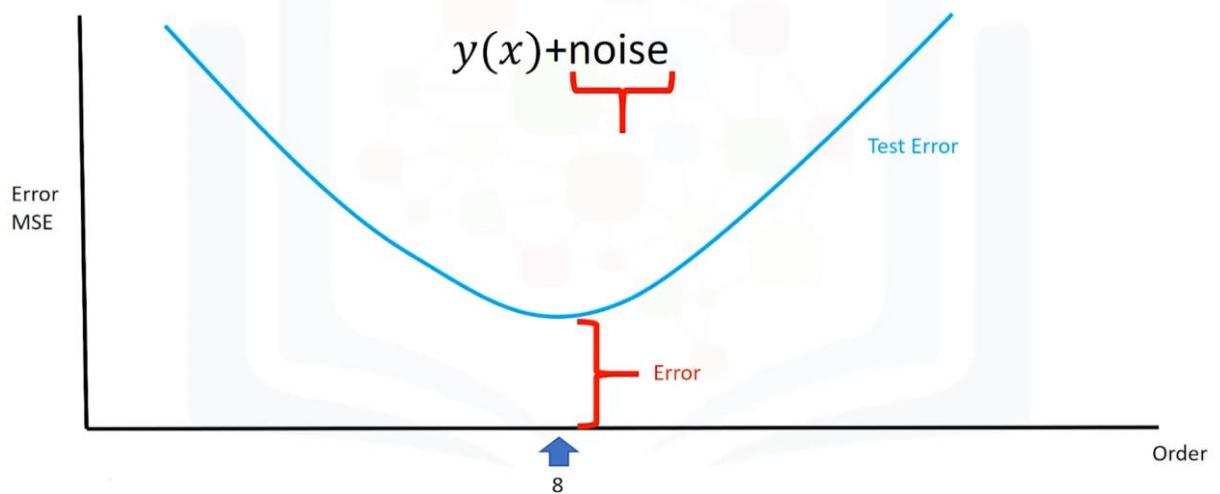
Существуют и другие источники ошибок. Например, наше предположение о полиноме может быть неверным.

Model Selection

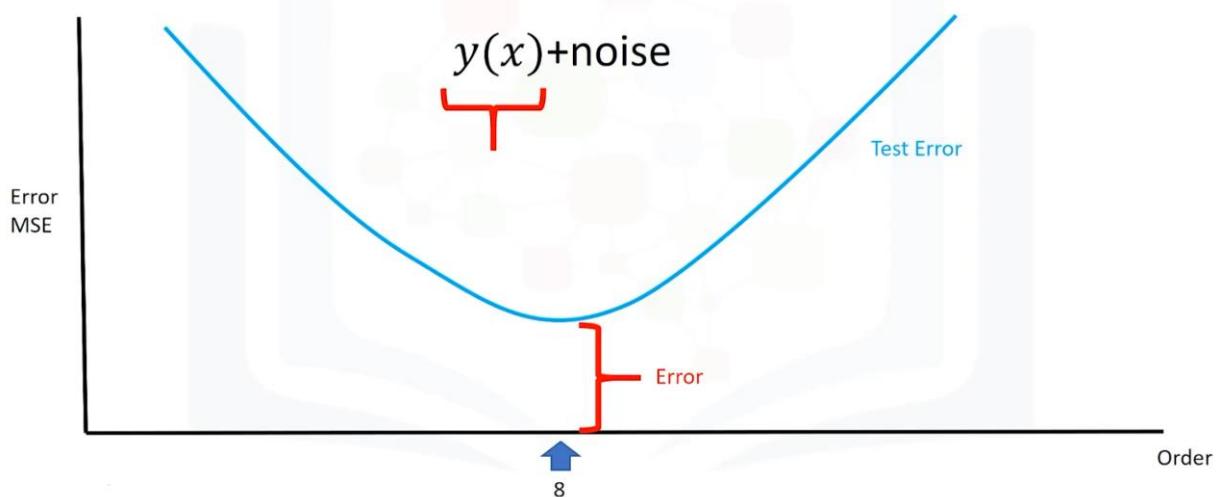


Наши точки выборки могли быть получены из другой функции. Например, на этом графике данные получены из синусоидальной волны.

Model Selection

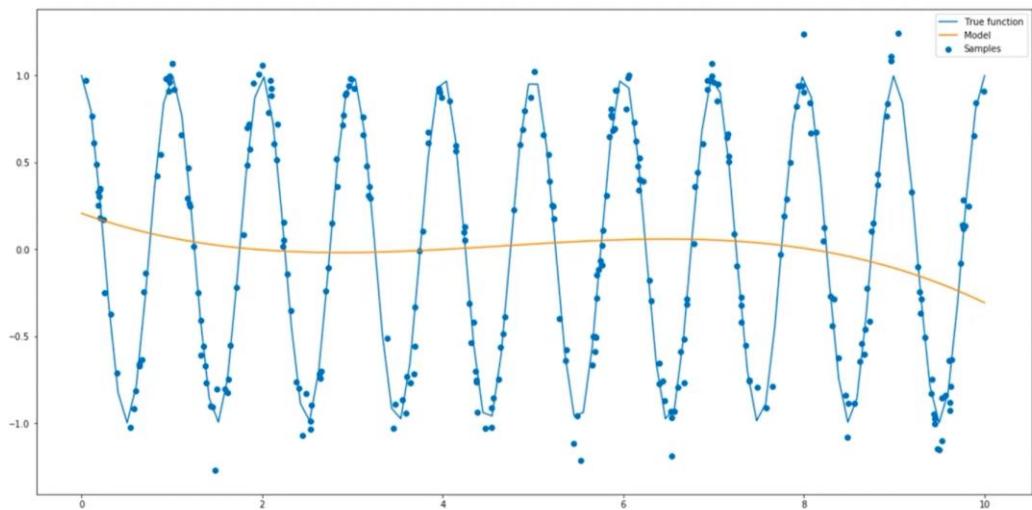


Model Selection



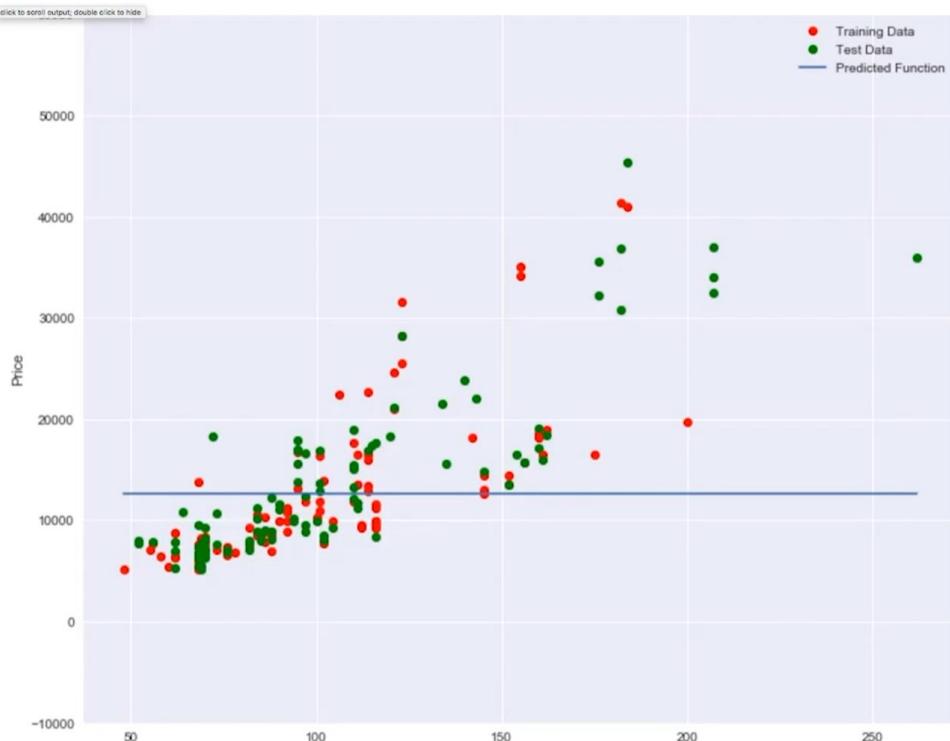
Полиномиальная функция не очень хорошо подходит для синусоиды. Для реальных данных модель может оказаться слишком сложной для подгонки или у нас может не быть данных нужного типа для оценки функции.

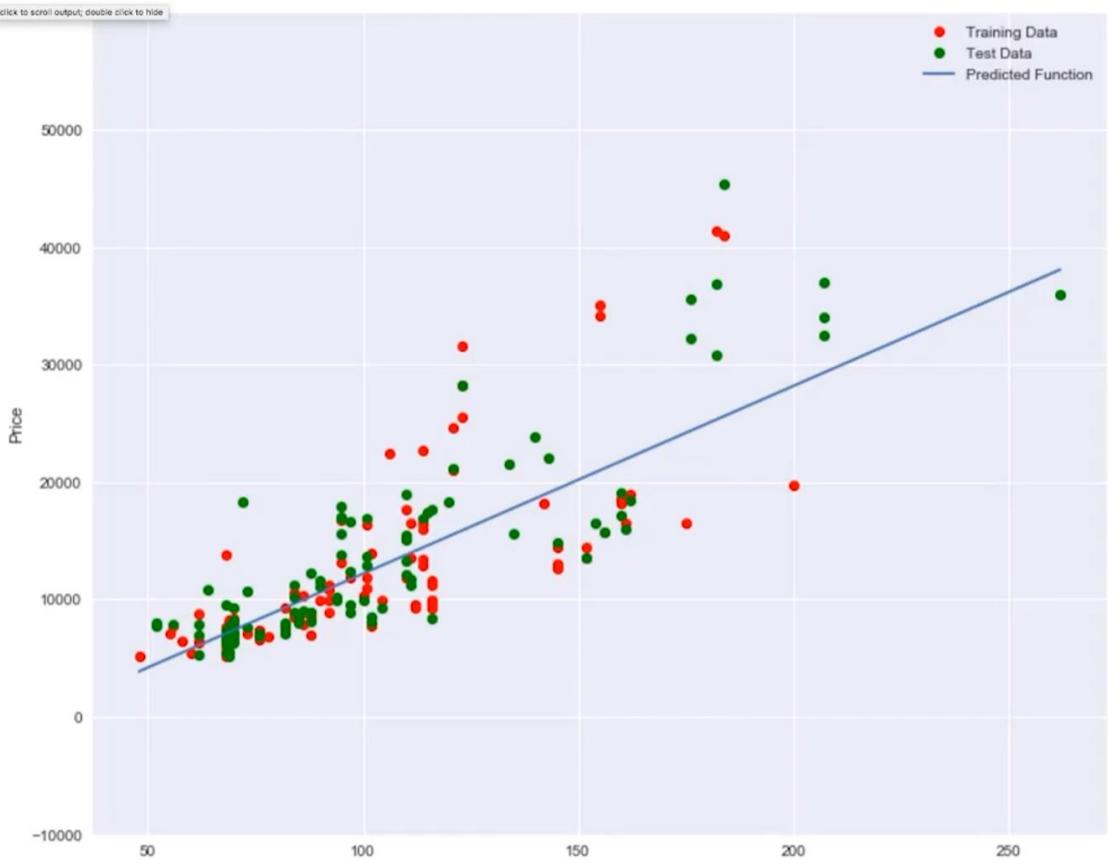
Model Selection



Давайте попробуем использовать полиномы разного порядка на реальных данных по лошадиной силе. Красные точки представляют собой обучающие данные.

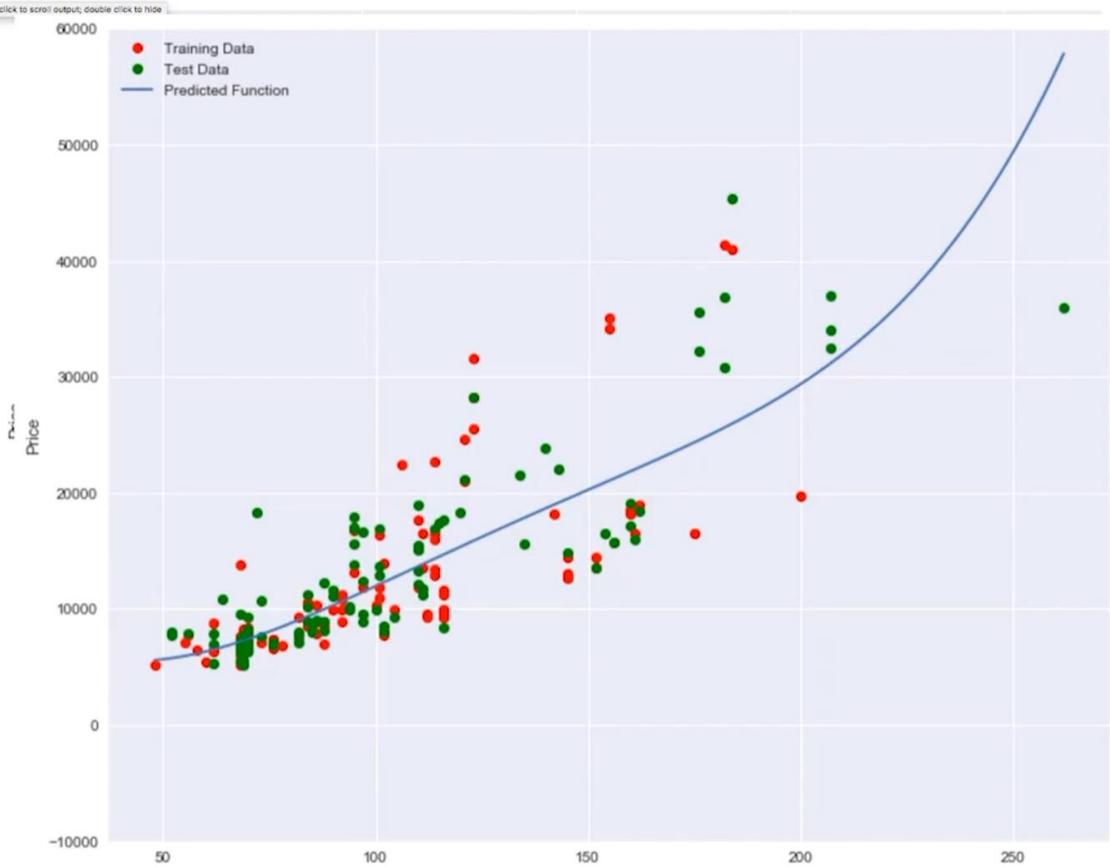
Зеленые точки - тестовые данные. Если мы просто используем среднее значение данных, наша модель работает плохо.

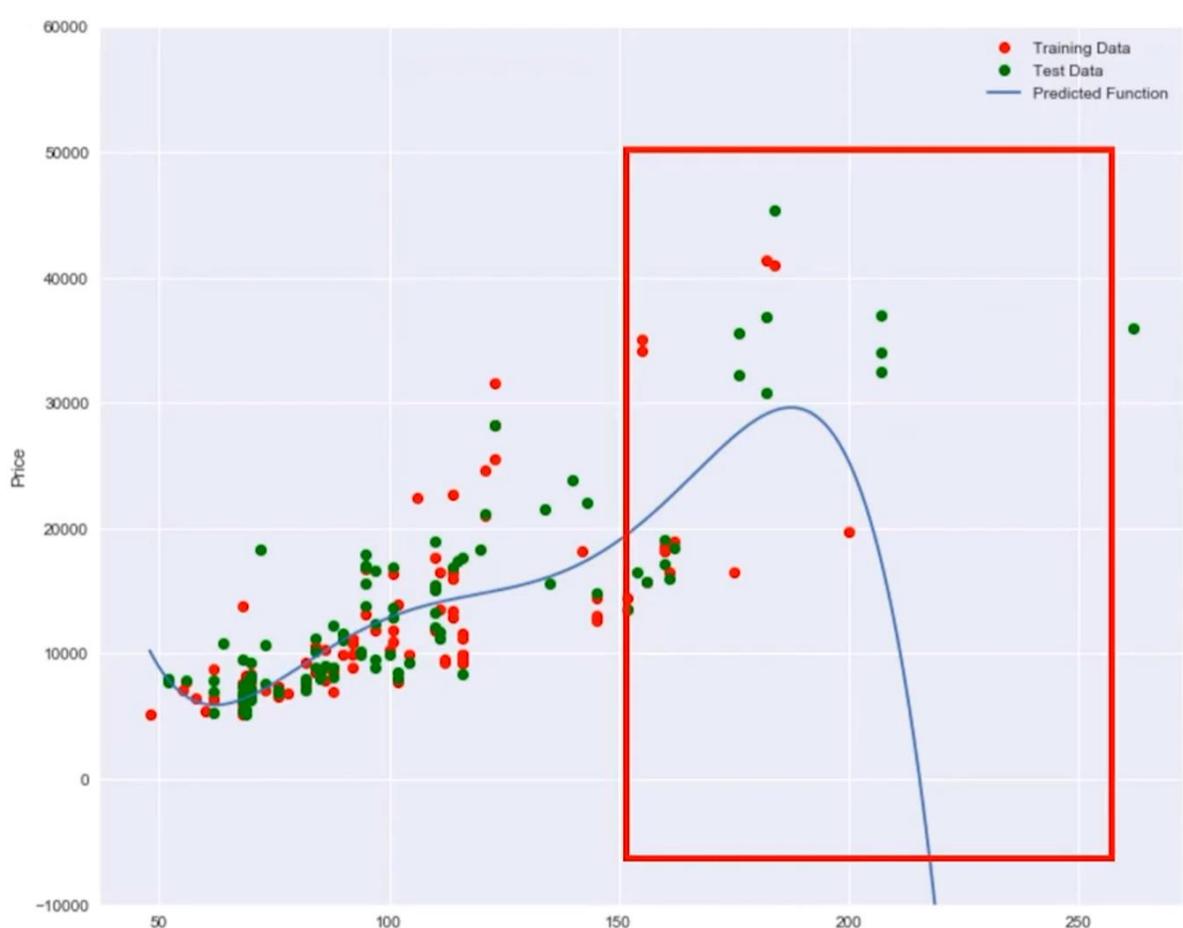




Линейная функция действительно лучше соответствует данным. Модель второго порядка похожа на линейную функцию.

Функция третьего порядка также кажется возрастающей, как и предыдущие два порядка. Здесь мы видим полином четвертого порядка. При мощности около 200 лошадиных сил, прогнозируемая цена внезапно снижается.



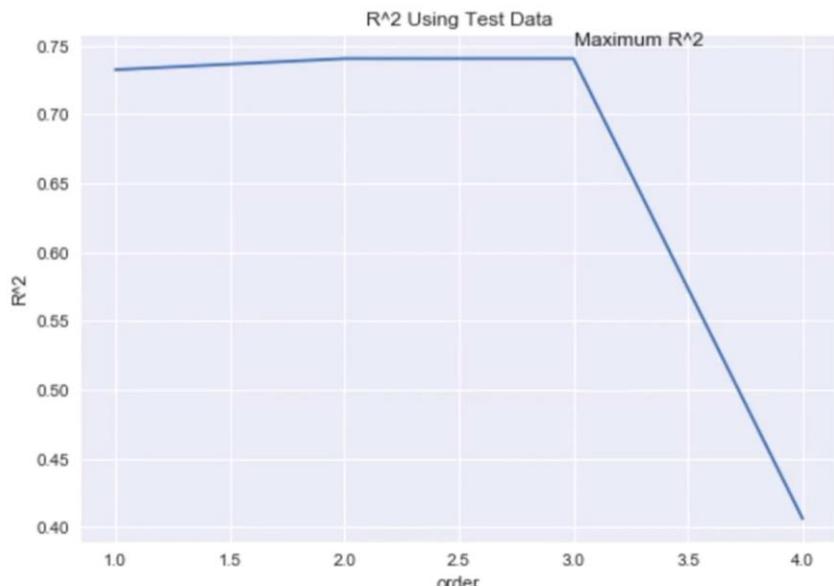


Это кажется ошибочным.

Давайте воспользуемся R-квадрат, чтобы проверить, верно ли наше предположение. Ниже показан график значения R-квадрат.

Горизонтальная ось представляет полиномиальные модели порядка. Чем ближе R-квадрат к единице, тем точнее модель.

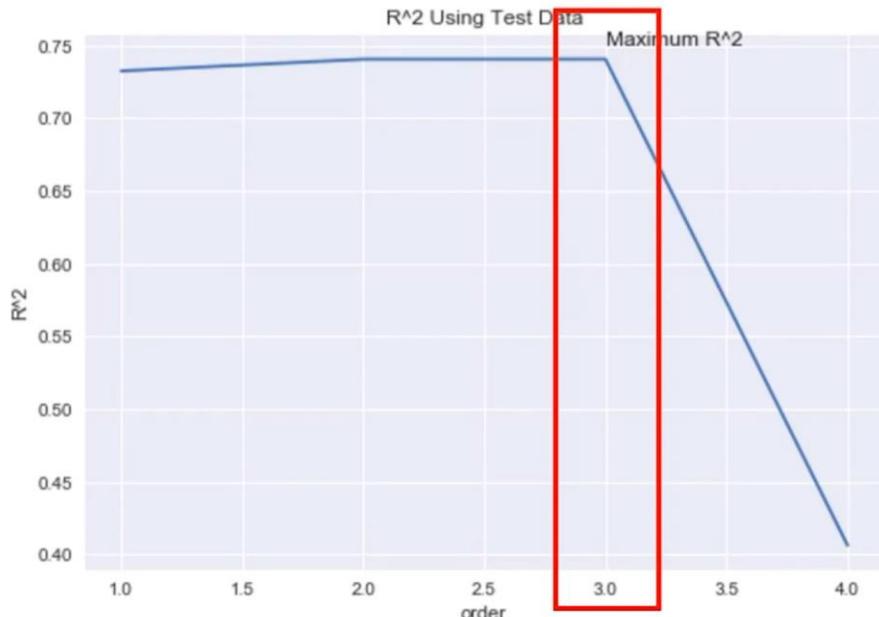
Model Selection



Здесь мы видим, что R-квадрат оптимален, когда порядок полинома равен трем.

При увеличении порядка до четырех R-квадрат резко уменьшается, подтверждая наше первоначальное предположение.

Model Selection



Мы можем рассчитать различные значения R-квадрат следующим образом.
Сначала мы создаем пустой список для хранения значений.

```
Rsqu_test=[]  
order=[1,2,3,4]  
for n in order:  
    pr=PolynomialFeatures(degree=n)  
    x_train_pr=pr.fit_transform(x_train[['horsepower']])  
    x_test_pr=pr.fit_transform(x_test[['horsepower']])  
    lr.fit(x_train_pr,y_train)  
    Rsqu_test.append(lr.score(x_test_pr,y_test))
```

Создаем список, содержащий различные порядки полиномов. Затем выполняем итерации по списку с помощью цикла.

Мы создаем объект полиномиального признака с порядком полинома в качестве параметра. Мы преобразуем обучающие и тестовые данные в полином с помощью метода fit transform. Мы подгоняем регрессионную модель, используя преобразованные данные. Затем мы вычисляем R-квадрат по тестовым данным и сохраняем его в массиве.

ГРЕБНЕВАЯ РЕГРЕССИЯ

Гребневая регрессия - это регрессия, которая используется в модели множественной регрессии, когда возникает мультиколлинеарность.

Мультиколлинеарность - это когда существует сильная связь между независимыми переменными.

Гребневая регрессия очень часто используется в полиномиальной регрессии. В следующем видеоролике показано, как регрессия гребня используется для регуляризации и уменьшения стандартных ошибок, чтобы избежать чрезмерной подгонки регрессионной модели.

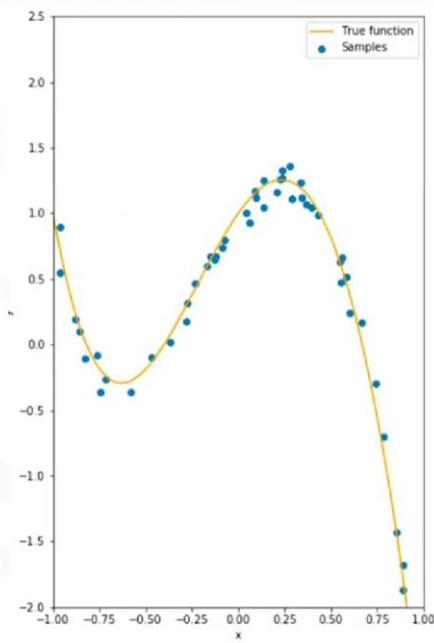
Гребневая регрессия предотвращает чрезмерную подгонку.

В этом видео мы сосредоточимся на полиномиальной регрессии для визуализации, но чрезмерная подгонка также является большой проблемой, когда у вас есть несколько независимых переменных, или характеристик.

Рассмотрим следующий полином четвертого порядка оранжевого цвета.

Ridge Regression

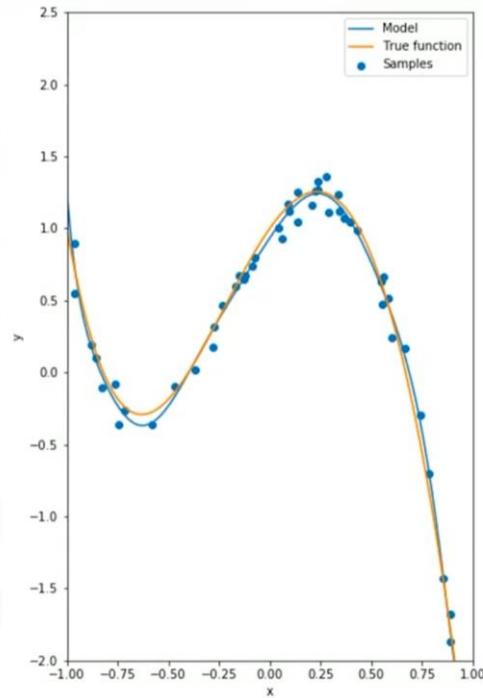
$$y = 1 + 2x - 3x^2 - 4x^3 + x^4$$



Синие точки генерируются на основе этой функции. Мы можем использовать полином десятого порядка для подгонки данных.

Ridge Regression

$$y = 1 + 2x - 3x^2 - 4x^3 + x^4$$

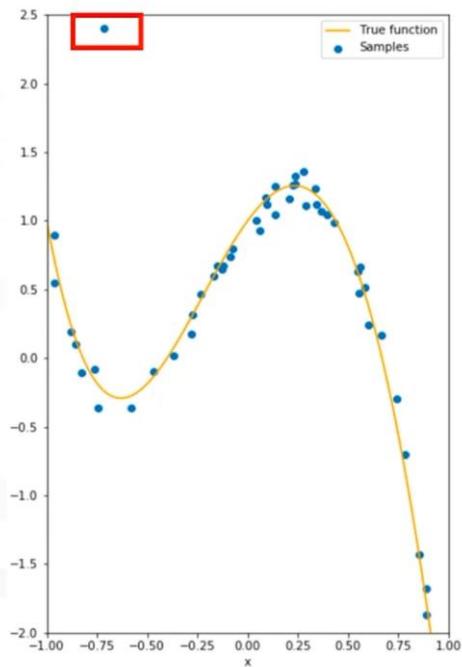


Оценочная функция синего цвета хорошо аппроксимирует истинную функцию.

Во многих случаях реальные данные содержат выбросы.

Ridge Regression

$$1 + 2x - 3x^2 - 4x^3 + x^4$$

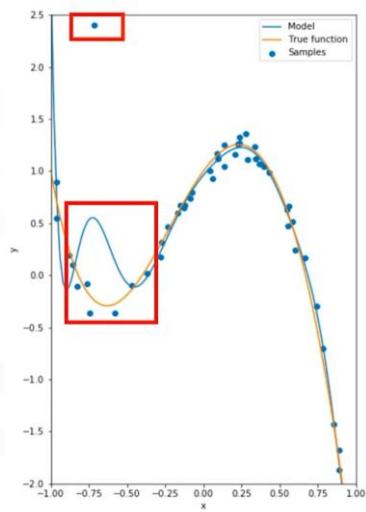


Например, эта

точка, показанная здесь, не похожа на функцию оранжевого цвета. Если мы используем полиномиальную функцию десятого порядка для подгонки данных, то расчетная функция, показанная синим цветом, будет неверной, и не является хорошей оценкой фактической функции в оранжевом цвете.

Ridge Regression

$$1 + 2x - 3x^2 - 4x^3 + x^4$$



Если мы рассмотрим выражение для расчетной функции, мы увидим, что коэффициенты оцениваемого полинома имеют очень большую величину.

Ridge Regression

$$\hat{y} = 1 + 2x - 3x^2 - 2x^3 - 12x^4 - 40x^5 + 80x^6 + 71x^7 - 141x^8 - 38x^9 + 75x^{10}$$

Это особенно заметно для полиномов высшего порядка.

Гребневая регрессия контролирует величину этих полиномиальных коэффициентов путем введения параметра альфа.

Ridge Regression

$$\hat{y} = 1 + 2x - 3x^2 - 2x^3 - 12x^4 - 40x^5 + 80x^6 + 71x^7 - 141x^8 - 38x^9 + 75x^{10}$$

Alpha
0
0.001
0.01
1
10

Альфа - это параметр, который мы выбираем перед подгонкой или обучением модели. Каждая строка в следующей таблице представляет собой возрастающее значение альфа. Давайте посмотрим, как различные значения альфа изменяют модель.

В этой таблице представлены коэффициенты полинома для различных значений альфа. Столбцы соответствуют различным коэффициентам полинома, а строки соответствуют различным значениям альфа.

Ridge Regression

$$\hat{y} = 1 + 2x - 3x^2 - 2x^3 - 12x^4 - 40x^5 + 80x^6 + 71x^7 - 141x^8 - 38x^9 + 75x^{10}$$

Alpha
0
0.001
0.01
1
10

x	x^2	x^3	x^4	x^5	x^6	x^7	x^8	x^9	x^{10}
2	-3	-2	-12	-40	80	71	-141	-38	75
2	-3	-7	5	4	-6	4	-4	4	6
1	-2	-5	-0.04	0.15	-1	1	-0.5	0.3	1
0.5	-1	-1	-0.614	0.70	-0.38	-0.56	-0.21	-0.5	-0.1
0	-0.5	-0.3	-0.37	-0.30	-0.30	-0.22	-0.22	-0.22	-0.17

По мере увеличения альфа параметры становятся меньше. Это наиболее очевидно для полиномиальных характеристик более высокого порядка. Однако альфа должна быть выбрана тщательно.

Ridge Regression

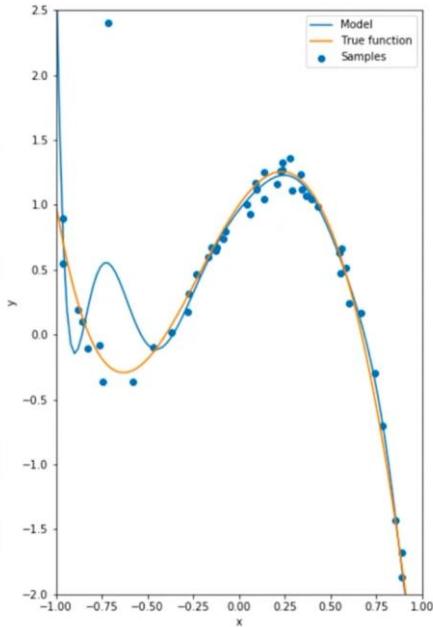
$$\hat{y} = 1 + 2x - 3x^2 - 2x^3 - 12x^4 - 40x^5 + 80x^6 + 71x^7 - 141x^8 - 38x^9 + 75x^{10}$$

<i>Alpha</i>	x	x^2	x^3	x^4	x^5	x^6	x^7	x^8	x^9	x^{10}
0	2	-3	-2	-12	-40	80	71	-141	-38	75
0.001	2	-3	-7	5	4	-6	4	-4	4	6
0.01	1	-2	-5	-0.04	0.15	-1	1	-0.5	0.3	1
1	0.5	-1	-1	-0.614	0.70	-0.38	-0.56	-0.21	-0.5	-0.1
10	0	-0.5	-0.3	-0.37	-0.30	-0.30	-0.22	-0.22	-0.22	-0.17

Если альфа слишком велика, коэффициенты приближаются к нулю и не будут соответствовать данным. Если альфа равна нулю, очевидна чрезмерная подгонка.

Ridge Regression

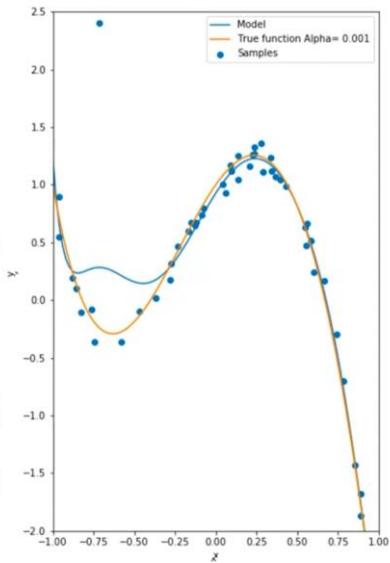
<i>alpha</i>
0
0.001
0.01
1
10



- При альфа, равном 0,001, избыточная подгонка начинает уменьшаться.

Ridge Regression

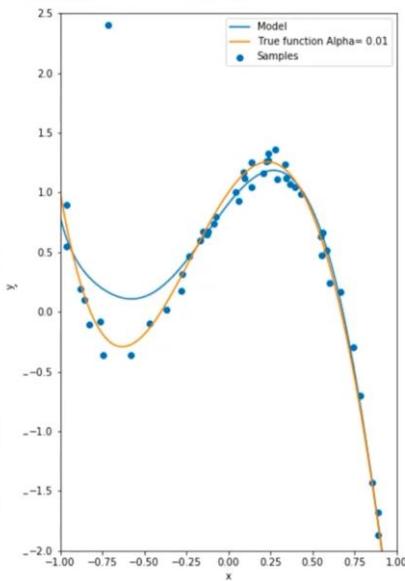
alpha
0
0.001
0.01
1
10



- При альфе, равной 0,01, расчетная функция совпадает с фактической.

Ridge Regression

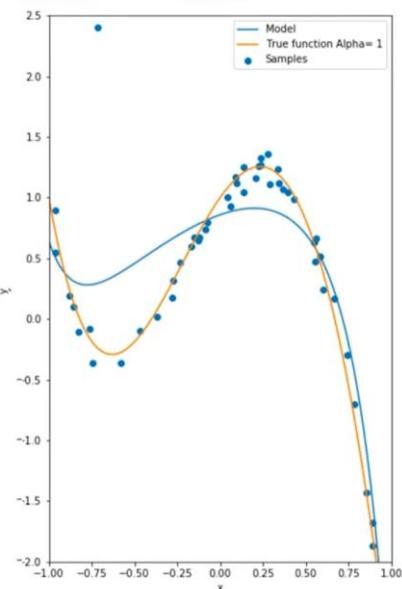
alpha
0
0.001
0.01
1
10



Когда альфа равна 1, мы видим первые признаки недостаточной подгонки.

Ridge Regression

alpha
0
0.001
0.01
1
10

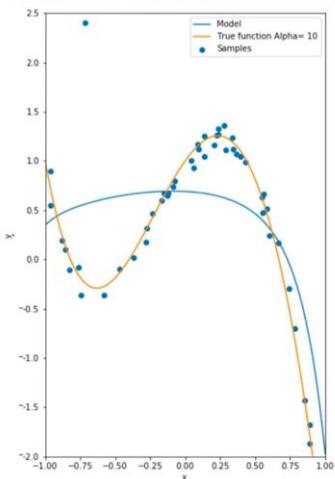


Оценочная функция не обладает достаточной гибкостью.

При альфа равном 10, мы видим экстремальное недоопределение. Она даже не отслеживает две точки.

Ridge Regression

alpha
0
0.001
0.01
1
10



Для того чтобы выбрать альфа, мы используем перекрестную валидацию.

Чтобы сделать предсказание с помощью гребневой регрессии, `import ridge from sklearn.linear_models.`

Ridge Regression

```
from sklearn.linear_model import Ridge
```

```
RidgeModel=Ridge(alpha=0.1)
```

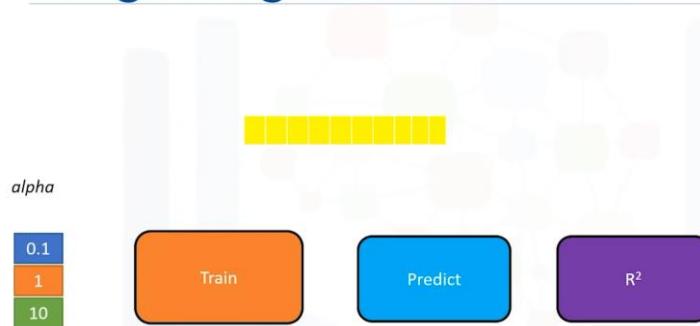
```
RidgeModel.fit(X,y)
```

```
Yhat=RidgeModel.predict(X)
```

- Создайте объект ridge с помощью конструктора.
- Параметр alpha является одним из аргументов конструктора.
- Мы обучаем модель с помощью метода fit.
- Чтобы сделать предсказание, мы используем метод predict.
- Для того чтобы определить параметр alpha, мы используем некоторые данные для обучения.
- Мы используем второй набор, называемый валидационными данными. Это похоже на тестовые данные, но он используется для выбора таких параметров, как альфа. Мы начинаем с небольшого значения альфа.
- Мы обучаем модель,
- делаем прогноз, используя данные валидации,
- затем вычисляем R-квадрат и сохраняем значения.

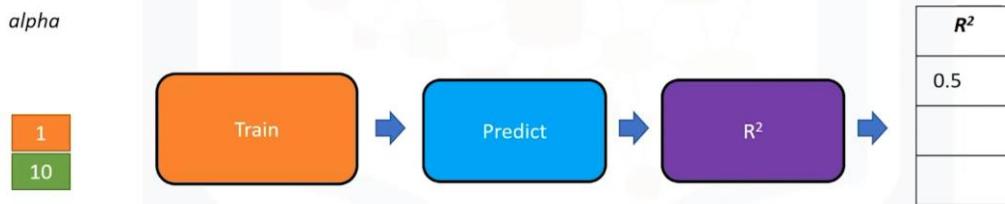
Повторите это действие для большего значения альфа.

Ridge Regression



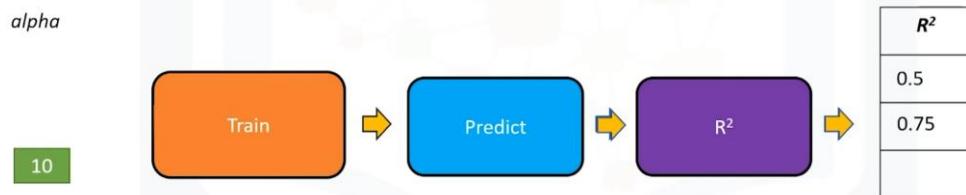
Мы обучаем модель, делаем прогноз, используя данные валидации, затем вычисляем R-квадрат и сохраняем значения.

Ridge Regression



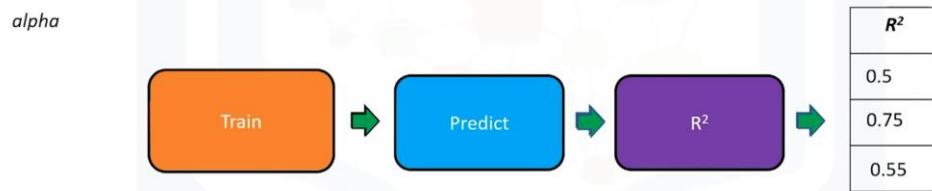
Повторяем значение для большего значения альфа. Мы снова обучаем модель, делаем предсказание, используя данные валидации, затем вычисляем R-квадрат и сохраняем значения R-квадрата.

Ridge Regression



Повторяя процесс для другого значения альфа, обучаем модель и делаем прогноз. Мы выбираем значение альфа, которое максимизирует R-квадрат.

Ridge Regression

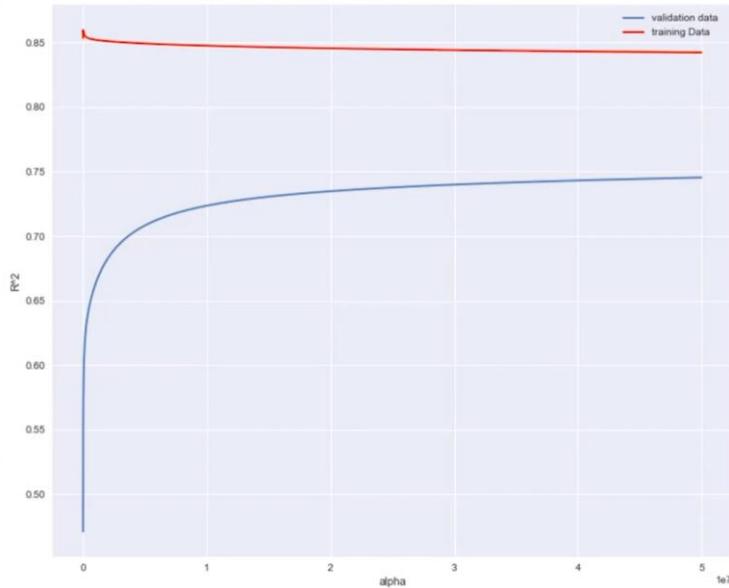


Обратите внимание, что мы можем использовать другие метрики для выбора значения альфа, например, среднюю квадратичную ошибку.

Проблема избыточной подгонки становится еще хуже, если у нас много признаков.

На следующем графике на вертикальной оси показаны различные значения R-квадрат.

Ridge Regression



Горизонтальная ось представляет собой различные значения альфа.

Мы используем несколько признаков из нашего набора данных о подержанных автомобилях и полиномиальную функцию второго порядка.

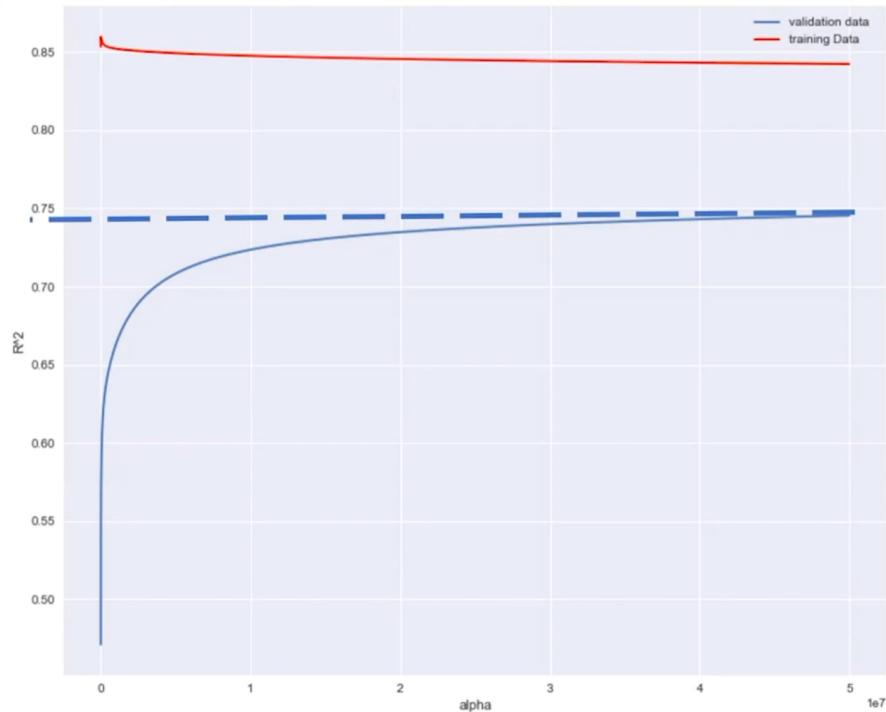
Данные для обучения выделены красным цветом, а данные для проверки - синим.

Мы видим, что по мере увеличения значения альфа, значение R-квадрат увеличивается и сходится примерно на 0,75.

В данном случае мы выбираем максимальное значение альфа, потому что проведение эксперимента при более высоких значениях альфа не оказывает

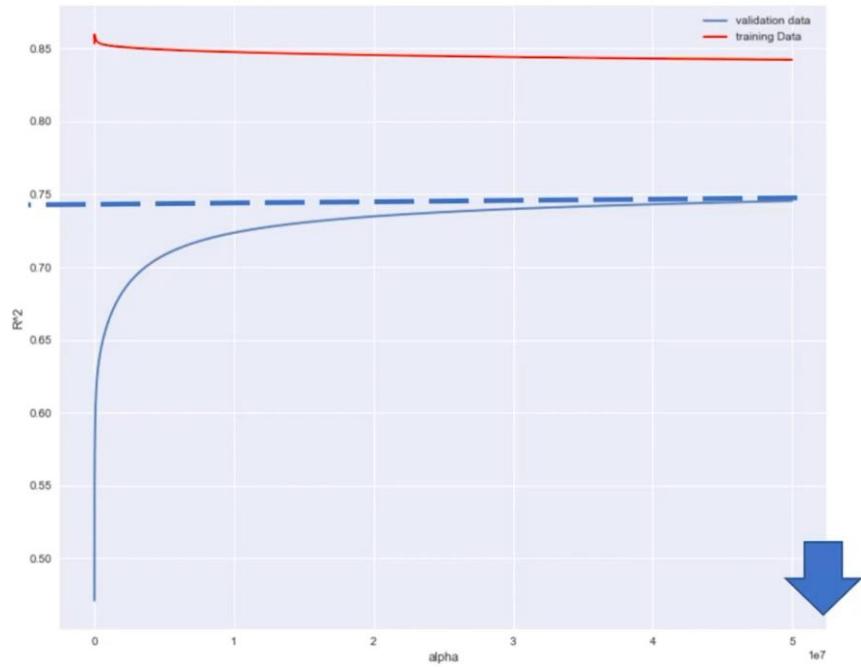
существенного влияния.

Ridge Regression



И наоборот, при увеличении альфа R -квадрат для тестовых данных уменьшается.

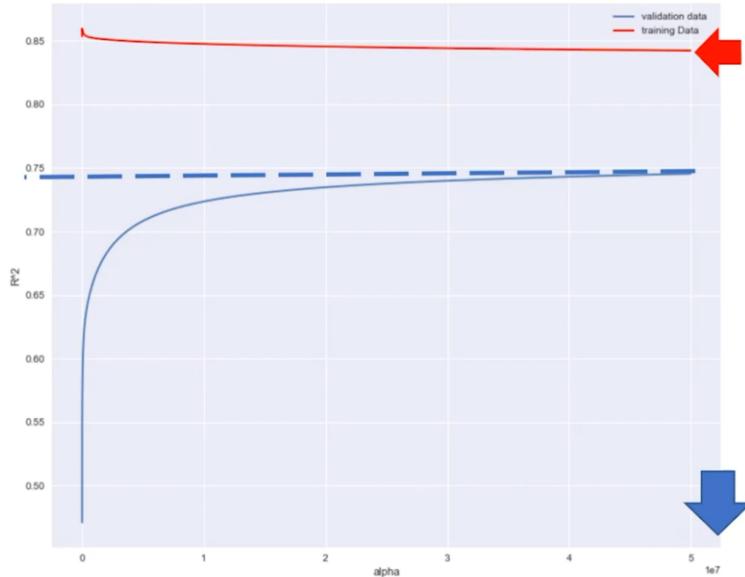
Ridge Regression



Это происходит потому, что альфа предотвращает чрезмерную подгонку. Это может улучшить результаты на нетестовых данных, но модель имеет худшую

производительность на тестовых данных.

Ridge Regression



Как построить этот график, смотрите в лабораторной работе.

GRID SEARCH

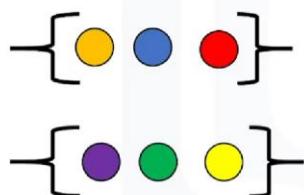
Grid Search позволяет нам просканировать множество свободных параметров с помощью нескольких строк кода.

Такие параметры, как альфа-термин, рассмотренный в предыдущем видео, не являются частью процесса подгонки или обучения.

Hyperparameters

- In the last section, the term alpha in Ridge regression is called a hyperparameter

hyperparamters



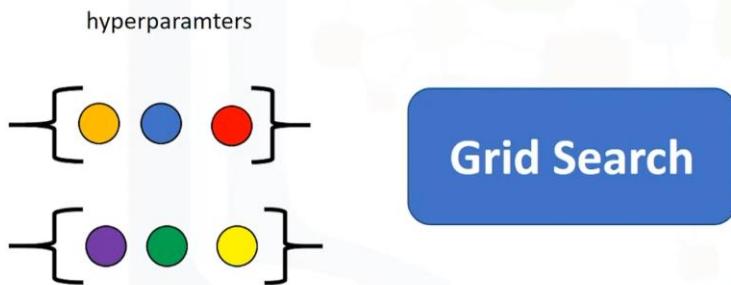
Grid Search

Эти значения называются гиперпараметрами.

Scikit-learn имеет средства для автоматического перебора этих гиперпараметров с помощью перекрестной проверки.

Hyperparameters

- In the last section, the term alpha in Ridge regression is called a hyperparameter
- Scikit-lean has a means of automatically iterating over these hyperparameters using cross-validation called Grid Search

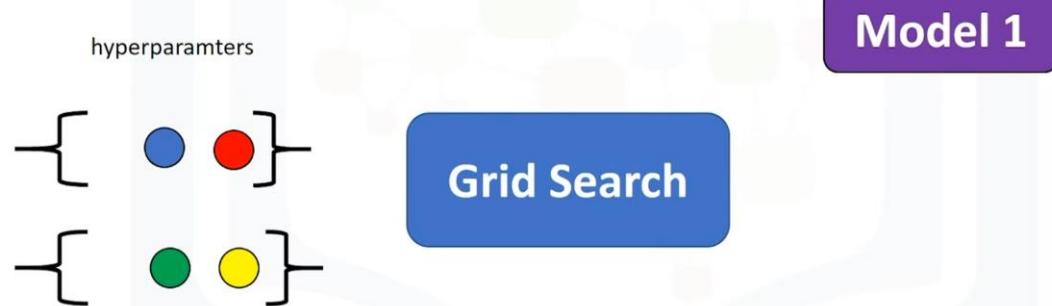


Этот метод называется Поиск по сетке.

При поиске по сетке берется модель или объекты, которые вы хотите и различные значения гиперпараметров.

Hyperparameters

- In the last section, the term alpha in Ridge regression is called a hyperparameter
- Scikit-lean has a means of automatically iterating over these hyperparameters using cross-validation called Grid Search



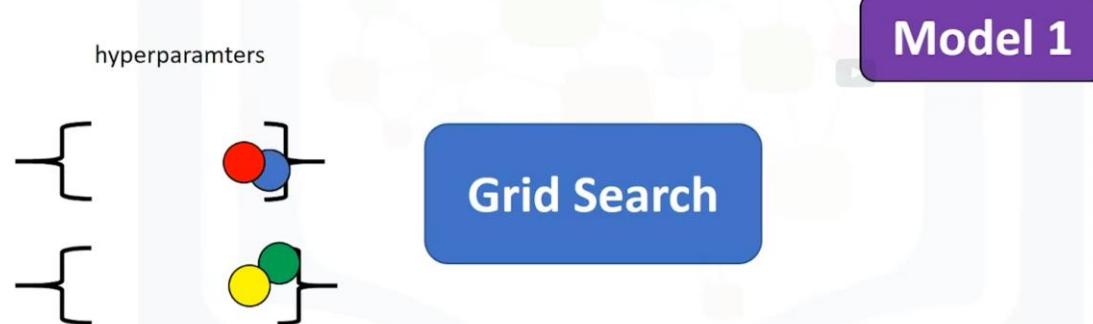
Затем вычисляется среднеквадратичная ошибка или R-квадрат для различных значений гиперпараметров, позволяя вам выбрать наилучшие значения.

Пусть маленькие кружки представляют различные гиперпараметры.

Мы начинаем с одного значения гиперпараметров и обучаем модель.

Hyperparameters

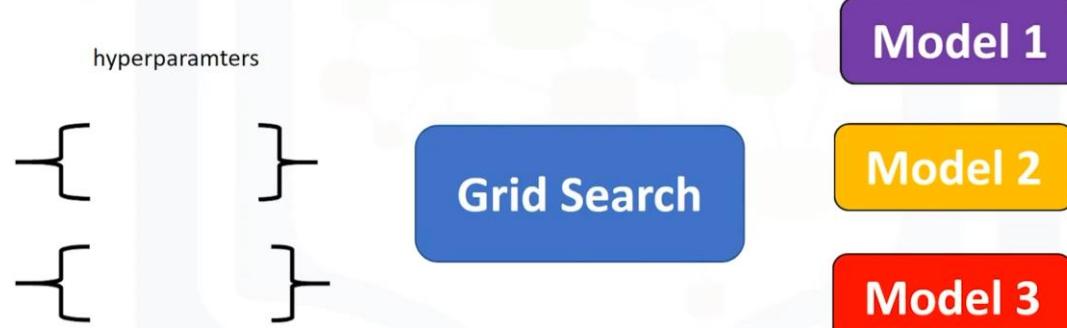
- In the last section, the term alpha in Ridge regression is called a hyperparameter
- Scikit-lean has a means of automatically iterating over these hyperparameters using cross-validation called Grid Search



Мы используем различные гиперпараметры для обучения модели.

Hyperparameters

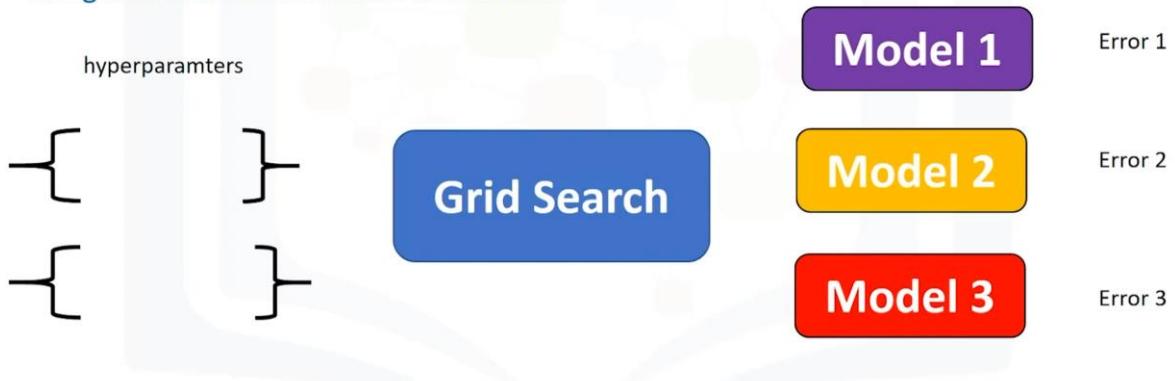
- In the last section, the term alpha in Ridge regression is called a hyperparameter
- Scikit-lean has a means of automatically iterating over these hyperparameters using cross-validation called Grid Search



Мы продолжаем этот процесс до тех пор, пока не исчерпаем различные свободные значения параметров. Каждая модель выдает ошибку.

Hyperparameters

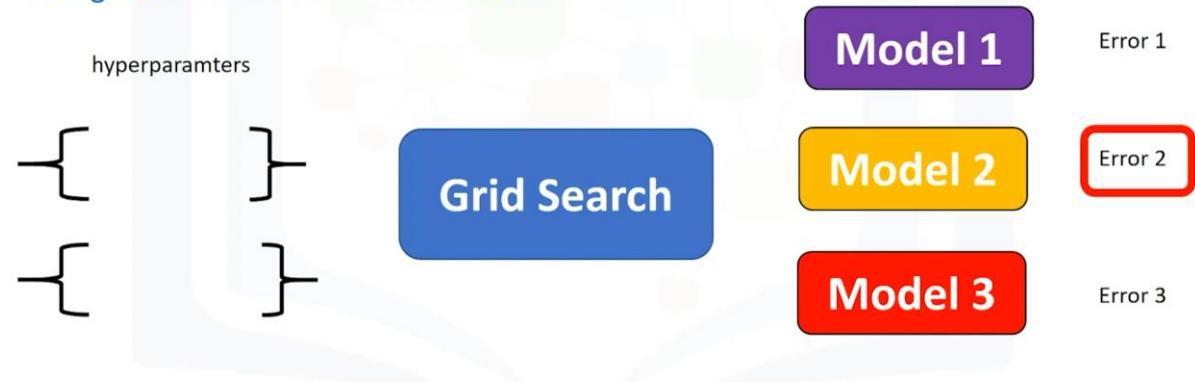
- In the last section, the term alpha in Ridge regression is called a hyperparameter
- Scikit-lean has a means of automatically iterating over these hyperparameters using cross-validation called Grid Search



Мы выбираем гиперпараметр, который минимизирует ошибку.

Hyperparameters

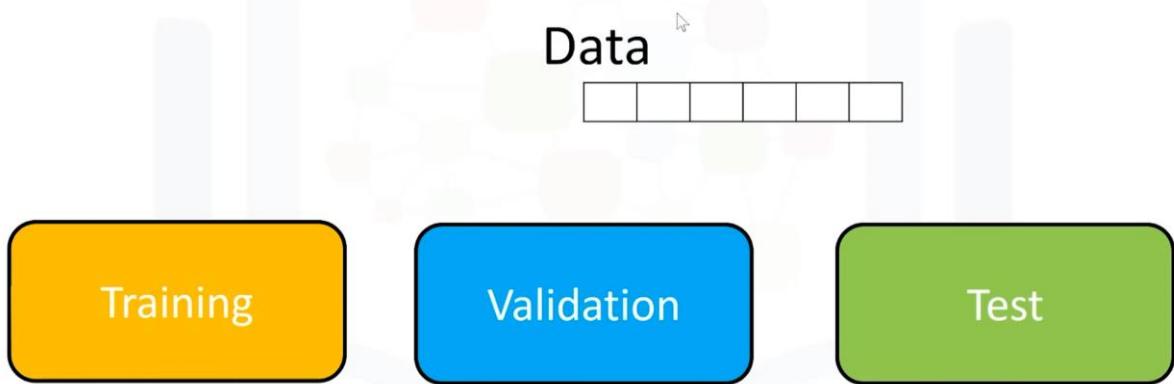
- In the last section, the term alpha in Ridge regression is called a hyperparameter
- Scikit-lean has a means of automatically iterating over these hyperparameters using cross-validation called Grid Search



Чтобы выбрать гиперпараметр, мы разделили наш набор данных на три части, обучающий набор, проверочный набор и тестовый набор.

Мы обучаем модель для различных гиперпараметров.

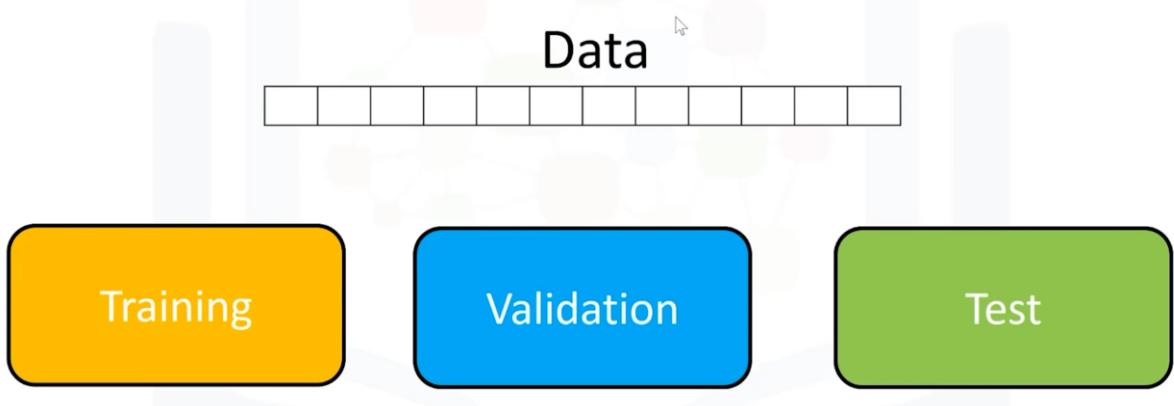
Grid Search



Мы используем R-квадрат или среднюю квадратичную ошибку для каждой модели.

Мы выбираем гиперпараметр, который минимизирует среднее квадратичное значение ошибки или максимизирует R-квадрат на проверочном множестве.

Grid Search



Наконец, мы проверяем производительность нашей модели на тестовых данных.

Это веб-страница scikit-learn, где указаны параметры конструктора объектов.

[Previous](#)
[sklearn.Ridge](#)
[Up](#)
[API Reference](#)scikit-learn v0.19.0
Other versionsPlease cite us if you use
the software.[sklearn.linear_model.Ridge](#)
Examples using
[sklearn.linear_model.Ridge](#)

sklearn.linear_model.Ridge

```
class sklearn.linear_model.Ridge(alpha=1.0, fit_intercept=True, normalize=False, copy_X=True, max_iter=None, tol=0.001, solver='auto', random_state=None)
```

[\[source\]](#)

Linear least squares with l2 regularization.



This model solves a regression model where the loss function is the linear least squares function and regularization is given by the l2-norm. Also known as Ridge Regression or Tikhonov regularization. This estimator has built-in support for multi-variate regression (i.e., when y is a 2d-array of shape [n_samples, n_targets]).

[Read more in the User Guide.](#)**Parameters:** `alpha` : {float, array-like}, shape (n_targets)

Regularization strength; must be a positive float. Regularization improves the conditioning of the problem and reduces the variance of the estimates. Larger values specify stronger regularization. Alpha corresponds to C^{-1} in other linear models such as LogisticRegression or LinearSVC. If an array is passed, penalties are assumed to be specific to the targets. Hence they must correspond in number.

fit_intercept : boolean

Whether to calculate the intercept for this model. If set to false, no intercept will be used in calculations (e.g. data is expected to be already centered).

normalize : boolean, optional, default False

This parameter is ignored when `fit_intercept` is set to False. If True, the regressors X will be normalized before regression by subtracting the mean and dividing by the l2-norm. If you wish to standardize, please use `sklearn.preprocessing.StandardScaler` before calling `fit` on an estimator with `normalize=False`.

copy_X : boolean, optional, default True

If True, X will be copied; else, it may be overwritten.

Следует отметить, что атрибуты объекта также называются параметрами.

Мы не будем делать различие, даже если некоторые из параметры не являются гиперпараметрами как таковыми.

В этом модуле мы сосредоточимся на гиперпараметре `alpha` и параметре нормализации.

Значение вашего Grid Search - это список Python, содержащий словарь Python.

Grid Search

```
parameters = [{ 'alpha' : [1, 10, 100, 1000] } ]
```

Ключ - это имя свободного параметра.

Значение словаря - это различные значения свободного параметра.

Это можно рассматривать как таблицу с различными значениями свободного параметра.

Grid Search

```
parameters = [ { 'alpha' : [1, 10, 100, 1000] } ]
```

Alpha	1	10	100	1000
-------	---	----	-----	------

У нас также есть объект или модель.

При поиске по сетке используется метод подсчета баллов.

Grid Search

```
parameters = [ { 'alpha' : [1, 10, 100, 1000] } ]
```

Alpha	1	10	100	1000
-------	---	----	-----	------

Ridge()

В данном случае R-квадрат - количество складок, модель или объект, и значения свободных параметров.

Grid Search

Ridge()

Scoring

Number
of Folds

Grid Search CV

Alpha	1	10	100	1000
-------	---	----	-----	------

Некоторые результаты включают различные оценки для разных значений свободных параметров.

В данном случае R-квадрат вместе со значениями свободных параметров, которые имеют наилучшую оценку.

Grid Search

Grid Search CV

Alpha	1	10	100	1000
R^2	0.74	0.35	0.073	0.008

Сначала мы импортируем необходимые нам библиотеки, включая GridSearchCV, словарь значений параметров.

```

from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV

parameters1= [{"alpha": [0.001,0.1,1, 10, 100, 1000,10000,100000,1000000]}]

RR=Ridge()

Grid1 = GridSearchCV(RR, parameters1, cv=4)

Grid1.fit(x_data[['horsepower', 'curb-weight', 'engine-size', 'highway-mpg']],y_data)

Grid1.best_estimator_

scores = Grid1.cv_results_
scores['mean_test_score']

```

- Мы создаем объект гребневой регрессии или модель.
- Затем мы создаем объект GridSearchCV.
- Входными данными являются объект гребневой регрессии, значения параметров и количество складок.

Мы будем использовать R-квадрат.

Это метод оценки по умолчанию. Мы подгоняем объект.

Мы можем найти лучшие значения для свободных параметров с помощью атрибута best estimator.

Мы также можем получить такую информацию, как средняя оценка на данных валидации с помощью атрибута CV result. Преимущества поиска по сетке в том, как быстро мы можем проверить несколько параметров.

Например, в гребневой регрессии есть возможность нормализации данных.

Чтобы узнать, как нормировать,смотрите четвертый модуль. Термин альфа является первым элементом в словаре.

- Второй элемент - это параметр нормализации.
- Ключ - это имя параметра.
- Значение - это различные варианты в данном случае, потому что мы можем либо нормализовать данные, либо нет.
- Значениями являются True или False соответственно.

Grid Search

```
parameters = [{ 'alpha' : [1, 10, 100, 1000], 'normalize' : [True, False] }]
```

•

Словарь - это таблица или сетка, содержащая два различных значения. Как и раньше, нам нужен объект или модель гребневой регрессии. Процедура аналогична, за исключением того, что у нас есть таблица или сетка с различными значениями параметров.

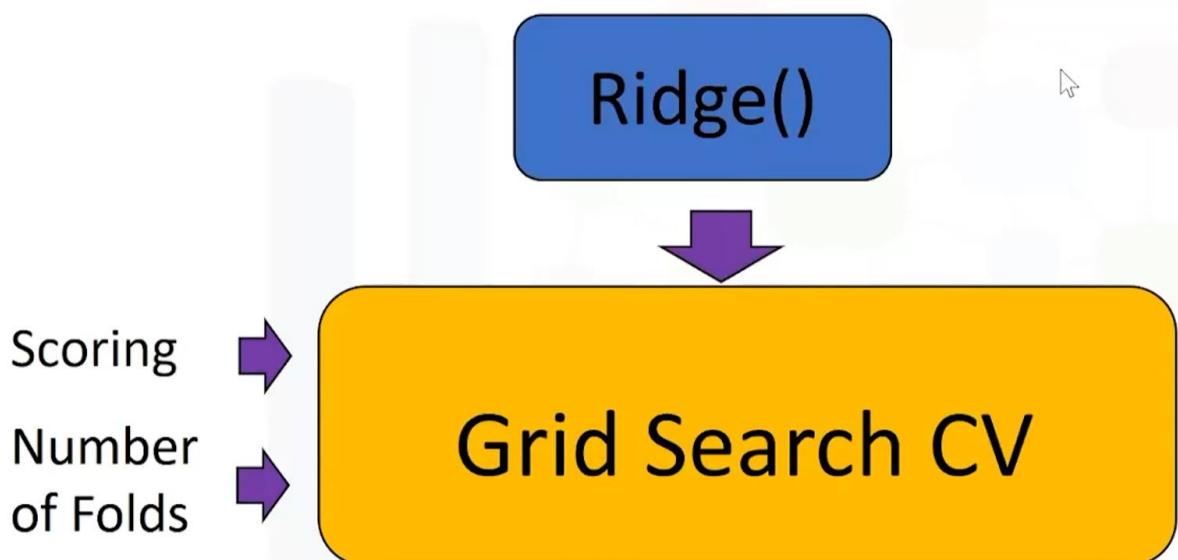
Grid Search

```
parameters = [ { 'alpha': [1, 10, 100, 1000], 'normalize': [True, False] } ]
```

Alpha	1	10	100	1000
Normalize	True	True	True	True
	False	False	False	False

На выходе мы получаем оценку для всех различных комбинаций значений параметров.

Grid Search

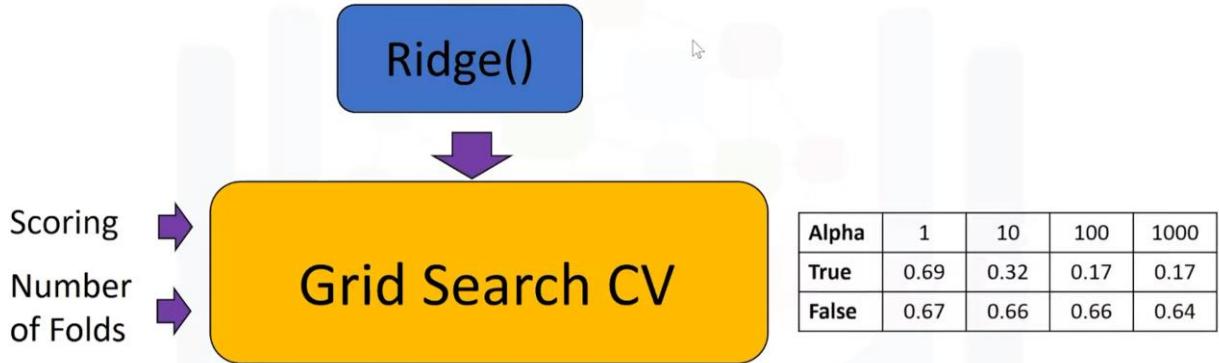


Alpha	1	10	100	1000
Normalize	True	True	True	True
	False	False	False	False

Код также аналогичен.

Словарь содержит различные свободные значения параметров. Мы можем найти наилучшее значение для свободных параметров.

Grid Search



Полученные оценки различных свободных параметров хранятся в этом словаре, Grid1.cv_results_.

```
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV

parameters2= [{"alpha": [0.001,0.1,1, 10, 100], 'normalize': [True, False] }]

RR=Ridge()

Grid1 = GridSearchCV(RR, parameters2, cv=4)

Grid1.fit(x_data[['horsepower', 'curb-weight', 'engine-size', 'highway-mpg']], y_data)

Grid1.best_estimator_

scores = Grid1.cv_results_
```

Мы можем распечатать оценки для различных значений свободных параметров.

Значения параметров хранятся, как показано здесь.

```
for param,mean_val, mean_test inzip(scores['params'],scores['mean_test_score'],scores['mean_train_score']):  
  
    print(param, "R^2 on test data:", mean_val, "R^2 on train data:",mean_test)
```

```
{'alpha': 0.001, 'normalize': True} R^2 on tesst data: 0.66605547293 R^2 on train data: 0.814001968709  
'alpha': 0.001, 'normalize': False} R^2 on tesst data: 0.665488366584 R^2 on train data: 0.814002698797  
'alpha': 0.1, 'normalize': True} R^2 on tesst data: 0.694175625356 R^2 on train data: 0.810546768311  
'alpha': 0.1, 'normalize': False} R^2 on tesst data: 0.665488937796 R^2 on train data: 0.814002698794  
'alpha': 1, 'normalize': True} R^2 on tesst data: 0.690486934584 R^2 on train data: 0.749104440368  
'alpha': 1, 'normalize': False} R^2 on tesst data: 0.665494127178 R^2 on train data: 0.814002698472  
'alpha': 10, 'normalize': True} R^2 on tesst data: 0.321376875232 R^2 on train data: 0.341856042902  
'alpha': 10, 'normalize': False} R^2 on tesst data: 0.665545680812 R^2 on train data: 0.8140026666  
'alpha': 100, 'normalize': True} R^2 on tesst data: 0.0170551710263 R^2 on train data: 0.0496044796826  
'alpha': 100, 'normalize': False} R^2 on tesst data: 0.666029359996 R^2 on train data: 0.813999791851  
'alpha': 1000, 'normalize': True} R^2 on tesst data: -0.0301961745066 R^2 on train data: 0.005184451599  
'alpha': 1000, 'normalize': False} R^2 on tesst data: 0.668968215369 R^2 on train data: 0.813870488264  
'alpha': 10000, 'normalize': True} R^2 on tesst data: -0.0351687400461 R^2 on train data: 0.000520784757979  
'alpha': 10000, 'normalize': False} R^2 on tesst data: 0.673346359342 R^2 on train data: 0.812583743226  
'alpha': 100000, 'normalize': True} R^2 on tesst data: -0.0356685844558 R^2 on train data: 5.2101975528e-05  
'alpha': 100000, 'normalize': False} R^2 on tesst data: 0.657818838432 R^2 on train data: 0.789541446486  
'alpha': 100000, 'normalize': True} R^2 on tesst data: -0.0356685844558 R^2 on train data: 5.2101975528e-05  
'alpha': 100000, 'normalize': False} R^2 on tesst data: 0.657818838432 R^2 on train data: 0.789541446486
```

В этом уроке вы узнали, как:

- Определять чрезмерную и недостаточную подгонку в прогностической модели: Переподгонка происходит, когда функция слишком близко подходит к точкам обучающих данных и захватывает шум данных. Недооптимизация относится к модели, которая не может смоделировать учебные данные или уловить тенденцию данных.
- Примените гребневую регрессию к моделям линейной регрессии: Гребневая регрессия – это регрессия, которая применяется в модели множественной регрессии, когда возникает мультиколлинеарность.
- Настроить гиперпараметры оценщика с помощью поиска по сетке: Поиск по сетке – это эффективная по времени техника настройки, которая исчерпывающе вычисляет оптимальные значения гиперпараметров, выполняемых для конкретных значений параметров оценщиков.