

Pandas

`pandas` – это библиотека для анализа, очистки и преобразования данных табличных данных в Python. Она предоставляет мощные для работы с табличными (DataFrame) и одномерными (Series) данными. Последнее используется, как правило, для временных рядов, а первое, в подавляющем числе случаев, для всего остального.

Установка: `pip install pandas`. Если при виде этого хочется спросить: "И что с этим делать?", то ответ будет в прошлой части.

Импорт:

```
In [1]: import pandas as pd  
import numpy as np
```

Series

Создадим простую последовательность.

```
In [2]: data = [10, 20, 30, 40, 50] # обычный список  
series = pd.Series(data)  
print(series)
```

```
0    10  
1    20  
2    30  
3    40  
4    50  
dtype: int64
```

Обратим внимание, что индексы были простираны автоматически.

О поддержке numpy многими библиотеками мы в прошлый раз говорили не зря.

```
In [3]: data = np.array(data) # массив numpy также поддерживается  
series = pd.Series(data)  
print(series)
```

```
0    10  
1    20  
2    30  
3    40  
4    50  
dtype: int64
```

Можем также сделать кастомную индексацию. Мы к ней ещё вернёмся позже.

```
In [4]: series = pd.Series([10, 20, 30], index=['a', 'b', 'c'])  
print(series)
```

```
a    10  
b    20  
c    30  
dtype: int64
```

DataFrame

Датафрейм --- по сути табличные данные, как в реляционной БД или Excel (или его аналоге).

Создадим его из словаря. Здесь ключи --- название столбцов, список в значениях --- строки каждого конкретного столбца.

```
In [5]: data = {  
    'Имя': ['Анна', 'Борис', 'Виктор'],  
    'Возраст': [25, 30, 22],  
    'Город': ['Москва', 'СПб', 'Казань']  
}  
df = pd.DataFrame(data)  
print(df)
```

	Имя	Возраст	Город
0	Анна	25	Москва
1	Борис	30	СПб
2	Виктор	22	Казань

Атрибуты DataFrame

```
In [6]: df.shape # размер
```

```
Out[6]: (3, 3)
```

```
In [7]: df.columns # Названия колонок
```

```
Out[7]: Index(['Имя', 'Возраст', 'Город'], dtype='object')
```

```
In [8]: print(df.index) # Индексы строк
```

```
RangeIndex(start=0, stop=3, step=1)
```

- `start=0` --- индекс начинается с 0,
- `stop=3` --- до трёх (не включая),
- `step=1` --- с шагом 1.

Загрузка и сохранение данных

Pandas поддерживает несколько источников данных: csv, Excel таблицы (и их заменители, аналогичные натуральному), базы данных и json. Наиболее распространённый (во всяком случае в некоммерческих ситуациях) --- это csv.

CSV (Comma-Separated Values, "значения, разделённые запятыми") --- это простой текстовый формат для хранения табличных данных. Каждая строка --- это одна

запись (строка таблицы), а значения (столбцы/ячейки) внутри неё (строки) разделяются запятыми, а также точками с запятой, табуляцией или другими разделителями.

Для прочтения файла необходимо знать какой именно разделитель используется. Можно просто открыть файл и посмотреть, можно сделать иначе.

Если вы используете линукс (иногда работает и на винде), то можете просто использовать `head`:

In [9]: `!head housing.csv`

```
Avg. Area Income,Avg. Area House Age,Avg. Area Number of Rooms,Avg. Area Number o
f Bedrooms,Area Population,Price,Address
79545.45857,5.682861322,7.009188143,4.09,23086.8005,1059033.558,"208 Michael Ferr
y Apt. 674
Laurabury, NE 37010-5101"
79248.64245,6.002899808,6.730821019,3.09,40173.07217,1505890.915,"188 Johnson Vie
ws Suite 079
Lake Kathleen, CA 48958"
61287.06718,5.86588984,8.51272743,5.13,36882.1594,1058987.988,"9127 Elizabeth Str
avenue
Danieltown, WI 06482-3489"
63345.24005,7.188236095,5.586728665,3.26,34310.24283,1260616.807,"USS Barnett
FPO AP 44820"
59982.19723,5.040554523,7.839387785,4.23,26354.10947,630943.4893,"USNS Raymond
```

Для Windows легче просто открыть и посмотреть файл. Но если есть нужда, то можно использовать следующий скрипт:

In [10]:

```
number_lines = 5
with open('housing.csv', 'r') as f:
    for _ in range(number_lines):
        print(f.readline().strip())
```

```
Avg. Area Income,Avg. Area House Age,Avg. Area Number of Rooms,Avg. Area Number o
f Bedrooms,Area Population,Price,Address
79545.45857,5.682861322,7.009188143,4.09,23086.8005,1059033.558,"208 Michael Ferr
y Apt. 674
Laurabury, NE 37010-5101"
79248.64245,6.002899808,6.730821019,3.09,40173.07217,1505890.915,"188 Johnson Vie
ws Suite 079
Lake Kathleen, CA 48958"
```

Здесь в конструкции `with` мы открываем для чтения (`'r'`) и закрываем файл после использования.

Как мы убедились, данный csv файл использует запятую в качестве разделителя, а также здесь нет столбца с индексами (в таком случае необходимо было бы либо указать, что данный столбец нужно использовать в качестве индекса; иногда когда данные заранее разбиты на несколько частей (например, обучение и тест) имеет смысл игнорировать столбец с индексами и переприсвоить им свои, т.к. в файлах могут быть индексы с пропусками).

Загрузка данных

Сделать датафрейм из csv файла легко.

```
In [11]: df = pd.read_csv('housing.csv')
df
```

Out[11]:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Avg. Area Population	Area	Price
0	79545.45857	5.682861	7.009188	4.09	23086.80050	1.059034e+06	208 M 674\r\nr
1	79248.64245	6.002900	6.730821	3.09	40173.07217	1.505891e+06	188 John 079\r\nnl
2	61287.06718	5.865890	8.512727	5.13	36882.15940	1.058988e+06	Stravenue
3	63345.24005	7.188236	5.586729	3.26	34310.24283	1.260617e+06	USS Ba
4	59982.19723	5.040555	7.839388	4.23	26354.10947	6.309435e+05	USNS R
...
4995	60567.94414	7.830362	6.137356	3.46	22837.36103	1.060194e+06	USNS '
4996	78491.27543	6.999135	6.576763	4.02	25616.11549	1.482618e+06	8489\r\nn
4997	63390.68689	7.250591	4.805081	2.13	33266.14549	1.030730e+06	4215 Tra 076\r\nJc
4998	68001.33124	5.534388	7.130144	5.44	42625.62016	1.198657e+06	USS Wa
4999	65510.58180	5.992305	6.792336	4.07	46501.28380	1.298950e+06	37778 Apt. 50

5000 rows × 7 columns

Сохранение данных

Если нам нужно сохранить датафрейм в файл, то сделать это также просто.

```
In [12]: df.to_csv('output.csv', index=False)
```

Основные операции с DataFrame

Обзор данных

head и tail

В функцию можно передать желаемое количество строк для вывода, по умолчанию выводит 5 строк. Обычно используется `head` для получения представления о данных, что там есть вообще.

In [13]: `df.head()`

Out[13]:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Avg. Area Population	Area	Price
0	79545.45857	5.682861	7.009188	4.09	23086.80050	1.059034e+06	208 Micha 674\r\nLa
1	79248.64245	6.002900	6.730821	3.09	40173.07217	1.505891e+06	188 Johnson 079\r\nLake
2	61287.06718	5.865890	8.512727	5.13	36882.15940	1.058988e+06	91 Stravenue\r\n
3	63345.24005	7.188236	5.586729	3.26	34310.24283	1.260617e+06	USS Barnet
4	59982.19723	5.040555	7.839388	4.23	26354.10947	6.309435e+05	USNS Raym

In [14]: `df.tail()`

Out[14]:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price	
4995	60567.94414	7.830362	6.137356	3.46	22837.36103	1060193.786	Williams AP 301
4996	78491.27543	6.999135	6.576763	4.02	25616.11549	1482617.729	PSC 92 8489\r\n429
4997	63390.68689	7.250591	4.805081	2.13	33266.14549	1030729.583	4215 Tracy 076\r\nJosf
4998	68001.33124	5.534388	7.130144	5.44	42625.62016	1198656.872	Wallace\r\n
4999	65510.58180	5.992305	6.792336	4.07	46501.28380	1298950.480	37778 Rid 509\r\nEa



info

Краткая информация: количество строк, типы данных, количество непустых значений:

In [15]: `df.info()`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 7 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   Avg. Area Income    5000 non-null   float64
 1   Avg. Area House Age 5000 non-null   float64
 2   Avg. Area Number of Rooms 5000 non-null   float64
 3   Avg. Area Number of Bedrooms 5000 non-null   float64
 4   Area Population     5000 non-null   float64
 5   Price               5000 non-null   float64
 6   Address             5000 non-null   object 
dtypes: float64(6), object(1)
memory usage: 273.6+ KB
```

describe

Это очень полезная функция, поскольку даёт много информации в компактном объёме. Отсюда можно узнать:

- Есть ли пропущенные значения (если `count` меньше, чем количество строк, убедимся на следующем датасете)
- Признаки с аномально высокими или низкими значениями (`min / max`)
- Признаки с перекошенным распределением (`mean` (среднее) далеко от 50% (медиана))
- Стандартное отклонение `std`, а также его сравнение между колонками (где больше стандартное отклонение, где меньше)
- Признаки с низкой дисперсией ($std \approx 0$) могут быть бесполезны для машинного обучения

In [16]: `df.describe()`

Out[16]:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Pri
count	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5.000000e+(0)
mean	68583.108984	5.977222	6.987792	3.981330	36163.516039	1.232073e+(0)
std	10657.991214	0.991456	1.005833	1.234137	9925.650114	3.531176e+(0)
min	17796.631190	2.644304	3.236194	2.000000	172.610686	1.593866e+(0)
25%	61480.562390	5.322283	6.299250	3.140000	29403.928700	9.975771e+(0)
50%	68804.286405	5.970429	7.002902	4.050000	36199.406690	1.232669e+(0)
75%	75783.338665	6.650808	7.665871	4.490000	42861.290770	1.471210e+(0)
max	107701.748400	9.519088	10.759588	6.500000	69621.713380	2.469066e+(0)

Обработка данных

Для этой секции нам нужен другой датасет. К сожалению, я не нашёл датасета с меньшим количеством столбцов, а создавать датасет самому казалось слишком искусственным и неинтересным. Главное не пугайтесь, что все столбцы не помещаются на экране.

In [17]: `file_name = 'AmesHousing.csv'`

In [18]: `number_lines = 5
with open(file_name, 'r') as f:
 for _ in range(number_lines):
 print(f.readline().strip())`

Order,PID,MS SubClass,MS Zoning,Lot Frontage,Lot Area,Street,Alley,Lot Shape,Land Contour,Utilities,Lot Config,Land Slope,Neighborhood,Condition 1,Condition 2,Bldg Type,House Style,Overall Qual,Overall Cond,Year Built,Year Remod/Add,Roof Style,Roof Matl,Exterior 1st,Exterior 2nd,Mas Vnr Type,Mas Vnr Area,Exter Qual,Exter Cond,Foundation,Bsmt Qual,Bsmt Cond,Bsmt Exposure,BsmtFin Type 1,BsmtFin SF 1,BsmtFin Type 2,BsmtFin SF 2,Bsmt Unf SF,Total Bsmt SF,Heating,Heating QC,Central Air,Electrical,1st Flr SF,2nd Flr SF,Low Qual Fin SF,Gr Liv Area,Bsmt Full Bath,Bsmt Half Bath,Full Bath,Half Bath,Bedroom AbvGr,Kitchen AbvGr,Kitchen Qual,TotRms AbvGrd,Functional,Fireplaces,Fireplace Qu,Garage Type,Garage Yr Blt,Garage Finish,Garage Cars,Garage Area,Garage Qual,Garage Cond,Paved Drive,Wood Deck SF,Open Porch SF,Enclosed Porch,3Ssn Porch,Screen Porch,Pool Area,Pool QC,Fence,Misc Feature,Misc Val,Mo Sold,Yr Sold,Sale Type,Sale Condition,SalePrice

1,0526301100,020,RL,141,31770,Pave,NA,IR1,Lvl,AllPub,Corner,Gtl,NAmes,Norm,Norm,1Fam,1Story,6,5,1960,1960,Hip,CompShg,BrkFace,Plywood,Stone,112,TA,TA,CBlock,TA,Gd,Gd,BLQ,639,Unf,0,441,1080,GasA,Fa,Y,SBrkr,1656,0,0,1656,1,0,1,0,3,1,TA,7,Typ,2,Gd,Attchd,1960,Fin,2,528,TA,TA,P,210,62,0,0,0,NA,NA,NA,0,5,2010,WD ,Normal,215000

2,0526350040,020,RH,80,11622,Pave,NA,Reg,Lvl,AllPub,Inside,Gtl,NAmes,Feedr,Norm,1Fam,1Story,5,6,1961,1961,Gable,CompShg,VinylSd,VinylSd,None,0,TA,TA,CBlock,TA,TA,No,Rec,468,LwQ,144,270,882,GasA,TA,Y,SBrkr,896,0,0,896,0,0,1,0,2,1,TA,5,Typ,0,NA,Attchd,1961,Unf,1,730,TA,TA,Y,140,0,0,0,120,0,NA,MnPrv,NA,0,6,2010,WD ,Normal,105000

3,0526351010,020,RL,81,14267,Pave,NA,IR1,Lvl,AllPub,Corner,Gtl,NAmes,Norm,Norm,1Fam,1Story,6,6,1958,1958,Hip,CompShg,Wd Sdng,Wd Sdng,BrkFace,108,TA,TA,CBlock,TA,T,A,No,ALQ,923,Unf,0,406,1329,GasA,TA,Y,SBrkr,1329,0,0,1329,0,0,1,1,3,1,Gd,6,Typ,0,NA,Attchd,1958,Unf,1,312,TA,TA,Y,393,36,0,0,0,0,NA,NA,Gar2,12500,6,2010,WD ,Normal,1,172000

4,0526353030,020,RL,93,11160,Pave,NA,Reg,Lvl,AllPub,Corner,Gtl,NAmes,Norm,Norm,1Fam,1Story,7,5,1968,1968,Hip,CompShg,BrkFace,BrkFace,None,0,Gd,TA,CBlock,TA,TA,No,ALQ,1065,Unf,0,1045,2110,GasA,Ex,Y,SBrkr,2110,0,0,2110,1,0,2,1,3,1,Ex,8,Typ,2,TA,Attchd,1968,Fin,2,522,TA,TA,Y,0,0,0,0,0,NA,NA,NA,0,4,2010,WD ,Normal,244000

Здесь у нас есть индекс (столбец `Order`), поэтому будем использовать его.

```
In [19]: df = pd.read_csv(file_name, index_col=0)
df.head()
```

Out[19]:

	PID	MS SubClass	MS Zoning	Lot Frontage	Lot Area	Street	Alley	Lot Shape	Land Contour
Order									
1	526301100	20	RL	141.0	31770	Pave	NaN	IR1	Lvl
2	526350040	20	RH	80.0	11622	Pave	NaN	Reg	Lvl
3	526351010	20	RL	81.0	14267	Pave	NaN	IR1	Lvl
4	526353030	20	RL	93.0	11160	Pave	NaN	Reg	Lvl
5	527105010	60	RL	74.0	13830	Pave	NaN	IR1	Lvl

5 rows × 81 columns

Удаление столбцов

Не все данные полезны для анализа. Например, в данном датасете есть `PID`, это уникальный ключ, но он никак не связан с какими-то объективными параметрами,

которые можно анализировать, поэтому нам его стоит удалить.

```
In [20]: df = df.drop('PID', axis=1)
df.head()
```

Out[20]:

	MS SubClass	MS Zoning	Lot Frontage	Lot Area	Street	Alley	Lot Shape	Land Contour	Utilities	Co
Order										
1	20	RL	141.0	31770	Pave	NaN	IR1	Lvl	AllPub	Co
2	20	RH	80.0	11622	Pave	NaN	Reg	Lvl	AllPub	In
3	20	RL	81.0	14267	Pave	NaN	IR1	Lvl	AllPub	Co
4	20	RL	93.0	11160	Pave	NaN	Reg	Lvl	AllPub	Co
5	60	RL	74.0	13830	Pave	NaN	IR1	Lvl	AllPub	In

5 rows × 80 columns

Добавить столбец

Добавить столбец также довольно просто. Можем присвоить всем столбцам какое-то значение по умолчанию или для каждой строки --- своё значение. Для этого нам понадобится список или массив. Суть не меняется.

```
In [21]: df['PID'] = -1
df.head()
```

Out[21]:

	MS SubClass	MS Zoning	Lot Frontage	Lot Area	Street	Alley	Lot Shape	Land Contour	Utilities	Co
Order										
1	20	RL	141.0	31770	Pave	NaN	IR1	Lvl	AllPub	Co
2	20	RH	80.0	11622	Pave	NaN	Reg	Lvl	AllPub	In
3	20	RL	81.0	14267	Pave	NaN	IR1	Lvl	AllPub	Co
4	20	RL	93.0	11160	Pave	NaN	Reg	Lvl	AllPub	Co
5	60	RL	74.0	13830	Pave	NaN	IR1	Lvl	AllPub	In

5 rows × 81 columns

Присвоенный столбец стал последним. Также возможно создать новый столбец/фичу как значение функции из других фич.

```
In [22]: new_feature = df['Lot Area']/df['Lot Frontage'] + 4
df.insert(0, 'New feature', new_feature) # Вставим на 0 место
df.head()
```

Out[22]:

	New feature	MS SubClass	MS Zoning	Lot Frontage	Lot Area	Street	Alley	Lot Shape	Land Contour
Order									
1	229.319149	20	RL	141.0	31770	Pave	NaN	IR1	Lvl
2	149.275000	20	RH	80.0	11622	Pave	NaN	Reg	Lvl
3	180.135802	20	RL	81.0	14267	Pave	NaN	IR1	Lvl
4	124.000000	20	RL	93.0	11160	Pave	NaN	Reg	Lvl
5	190.891892	60	RL	74.0	13830	Pave	NaN	IR1	Lvl

5 rows × 82 columns

Это можно было бы сделать проще:

```
{python}
df['New feature'] = df['Lot Area']/df['Lot Frontage'] + 4
```

но мне захотелось продемонстрировать функцию `insert`.

Новые столбцы нам не нужны, так что удалим их.

In [23]: `df = df.drop(['PID', 'New feature'], axis=1)`

Переименование столбцов

При необходимости можно переименовать желаемые столбцы. Для этого нужно передать в функцию словарь `{'Старое имя': 'Новое имя'}`. В словаре может быть больше одного вхождения.

In [24]: `df.rename(columns={'Street': 'Улица'}).head()`

	MS SubClass	MS Zoning	Lot Frontage	Lot Area	Улица	Alley	Lot Shape	Land Contour	Utilities Co
Order									
1	20	RL	141.0	31770	Pave	NaN	IR1	Lvl	AllPub Cc
2	20	RH	80.0	11622	Pave	NaN	Reg	Lvl	AllPub Ir
3	20	RL	81.0	14267	Pave	NaN	IR1	Lvl	AllPub Cc
4	20	RL	93.0	11160	Pave	NaN	Reg	Lvl	AllPub Cc
5	60	RL	74.0	13830	Pave	NaN	IR1	Lvl	AllPub Ir

5 rows × 80 columns

Также можно менять колонки местами. Пример ниже не блещет "питоничностью" (от `pythonic`), но в лоб показывает как поменять два столбца местами. Это не так

часто необходимо, но при желании можете найти более удобный способ.

```
In [25]: columns = list(df.columns)
f_col = columns[0]
columns[0] = columns[1]
columns[1] = f_col
df = df[columns]
df.head()
```

Out[25]:

	MS Zoning	MS SubClass	Lot Frontage	Lot Area	Street	Alley	Lot Shape	Land Contour	Utilities	Co
Order										
1	RL	20	141.0	31770	Pave	NaN	IR1	Lvl	AllPub	Co
2	RH	20	80.0	11622	Pave	NaN	Reg	Lvl	AllPub	In
3	RL	20	81.0	14267	Pave	NaN	IR1	Lvl	AllPub	Co
4	RL	20	93.0	11160	Pave	NaN	Reg	Lvl	AllPub	Co
5	RL	60	74.0	13830	Pave	NaN	IR1	Lvl	AllPub	In

5 rows × 80 columns

Пропущенные значения

Первый способ. Сравнение размеров и информации из `describe`.

Информация не помещается полностью, поэтому в VS Code можно открыть либо как прокручиваемый элемент, либо в отдельной вкладке.

```
In [26]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 2930 entries, 1 to 2930
Data columns (total 80 columns):
 #   Column            Non-Null Count Dtype  
 --- 
 0   MS Zoning         2930 non-null  object  
 1   MS SubClass        2930 non-null  int64   
 2   Lot Frontage       2440 non-null  float64 
 3   Lot Area           2930 non-null  int64   
 4   Street             2930 non-null  object  
 5   Alley              198 non-null   object  
 6   Lot Shape          2930 non-null  object  
 7   Land Contour       2930 non-null  object  
 8   Utilities          2930 non-null  object  
 9   Lot Config          2930 non-null  object  
 10  Land Slope          2930 non-null  object  
 11  Neighborhood        2930 non-null  object  
 12  Condition 1        2930 non-null  object  
 13  Condition 2        2930 non-null  object  
 14  Bldg Type          2930 non-null  object  
 15  House Style         2930 non-null  object  
 16  Overall Qual       2930 non-null  int64   
 17  Overall Cond        2930 non-null  int64   
 18  Year Built          2930 non-null  int64   
 19  Year Remod/Add      2930 non-null  int64   
 20  Roof Style          2930 non-null  object  
 21  Roof Matl           2930 non-null  object  
 22  Exterior 1st        2930 non-null  object  
 23  Exterior 2nd        2930 non-null  object  
 24  Mas Vnr Type        1155 non-null  object  
 25  Mas Vnr Area        2907 non-null  float64 
 26  Exter Qual          2930 non-null  object  
 27  Exter Cond          2930 non-null  object  
 28  Foundation          2930 non-null  object  
 29  Bsmt Qual           2850 non-null  object  
 30  Bsmt Cond           2850 non-null  object  
 31  Bsmt Exposure       2847 non-null  object  
 32  BsmtFin Type 1      2850 non-null  object  
 33  BsmtFin SF 1         2929 non-null  float64 
 34  BsmtFin Type 2      2849 non-null  object  
 35  BsmtFin SF 2         2929 non-null  float64 
 36  Bsmt Unf SF          2929 non-null  float64 
 37  Total Bsmt SF        2929 non-null  float64 
 38  Heating              2930 non-null  object  
 39  Heating QC            2930 non-null  object  
 40  Central Air           2930 non-null  object  
 41  Electrical            2929 non-null  object  
 42  1st Flr SF            2930 non-null  int64   
 43  2nd Flr SF            2930 non-null  int64   
 44  Low Qual Fin SF       2930 non-null  int64   
 45  Gr Liv Area           2930 non-null  int64   
 46  Bsmt Full Bath        2928 non-null  float64 
 47  Bsmt Half Bath        2928 non-null  float64 
 48  Full Bath             2930 non-null  int64   
 49  Half Bath              2930 non-null  int64   
 50  Bedroom AbvGr          2930 non-null  int64   
 51  Kitchen AbvGr          2930 non-null  int64   
 52  Kitchen Qual           2930 non-null  object  
 53  TotRms AbvGrd          2930 non-null  int64   
 54  Functional             2930 non-null  object
```

```
55 Fireplaces          2930 non-null   int64
56 Fireplace Qu        1508 non-null   object
57 Garage Type         2773 non-null   object
58 Garage Yr Blt      2771 non-null   float64
59 Garage Finish       2771 non-null   object
60 Garage Cars         2929 non-null   float64
61 Garage Area         2929 non-null   float64
62 Garage Qual         2771 non-null   object
63 Garage Cond         2771 non-null   object
64 Paved Drive         2930 non-null   object
65 Wood Deck SF        2930 non-null   int64
66 Open Porch SF       2930 non-null   int64
67 Enclosed Porch      2930 non-null   int64
68 3Ssn Porch          2930 non-null   int64
69 Screen Porch         2930 non-null   int64
70 Pool Area           2930 non-null   int64
71 Pool QC              13 non-null    object
72 Fence                572 non-null   object
73 Misc Feature         106 non-null   object
74 Misc Val             2930 non-null   int64
75 Mo Sold              2930 non-null   int64
76 Yr Sold              2930 non-null   int64
77 Sale Type            2930 non-null   object
78 Sale Condition        2930 non-null   object
79 SalePrice            2930 non-null   int64
dtypes: float64(11), int64(26), object(43)
memory usage: 1.8+ MB
```

In [27]: `df.describe()`

	MS SubClass	Lot Frontage	Lot Area	Overall Qual	Overall Cond	Year Built
count	2930.000000	2440.000000	2930.000000	2930.000000	2930.000000	2930.000000
mean	57.387372	69.224590	10147.921843	6.094881	5.563140	1971.356314
std	42.638025	23.365335	7880.017759	1.411026	1.111537	30.245361
min	20.000000	21.000000	1300.000000	1.000000	1.000000	1872.000000
25%	20.000000	58.000000	7440.250000	5.000000	5.000000	1954.000000
50%	50.000000	68.000000	9436.500000	6.000000	5.000000	1973.000000
75%	70.000000	80.000000	11555.250000	7.000000	6.000000	2001.000000
max	190.000000	313.000000	215245.000000	10.000000	9.000000	2010.000000

8 rows × 37 columns



Сравнив количество строк из `info` и количество непустых значений в каждом из столбцов можно понять, в каких столбцах есть пропуски.

Второй способ. `isna` и `isnull` --- это функции синонимы, они ничем не отличаются (вторая просто по аналогии с SQL), обе переводят значения в булевы, если значение отсутствует (NA), то `isna` возвращает `True`, в противном случае --- `False`.

```
In [28]: df.isna()
```

Out[28]:

Order	MS Zoning	MS SubClass	Lot Frontage	Lot Area	Street	Alley	Lot Shape	Land Contour	Utilities	I Con
1	False	False	False	False	False	True	False	False	False	False
2	False	False	False	False	False	True	False	False	False	False
3	False	False	False	False	False	True	False	False	False	False
4	False	False	False	False	False	True	False	False	False	False
5	False	False	False	False	False	True	False	False	False	False
...
2926	False	False	False	False	False	True	False	False	False	False
2927	False	False	True	False	False	True	False	False	False	False
2928	False	False	False	False	False	True	False	False	False	False
2929	False	False	False	False	False	True	False	False	False	False
2930	False	False	False	False	False	True	False	False	False	False

2930 rows × 80 columns



Как видим, в столбцах Alley , Pool QC и других есть пропущенные данные.

```
In [29]: pd.set_option('display.max_rows', None) # отключаем ограничение у самого pd  
# но при этом остаётся ограничение  
# в Jupyter  
df.isna().sum().sort_values(ascending=False)[lambda x: x > 0] # сортируем по убыванию
```

```
Out[29]: Pool QC      2917  
Misc Feature     2824  
Alley          2732  
Fence          2358  
Mas Vnr Type    1775  
Fireplace Qu     1422  
Lot Frontage     490  
Garage Yr Blt     159  
Garage Qual      159  
Garage Cond      159  
Garage Finish     159  
Garage Type       157  
Bsmt Exposure     83  
BsmtFin Type 2    81  
Bsmt Qual        80  
BsmtFin Type 1    80  
Bsmt Cond        80  
Mas Vnr Area       23  
Bsmt Full Bath     2  
Bsmt Half Bath     2  
BsmtFin SF 2       1  
Garage Area        1  
Total Bsmt SF       1  
BsmtFin SF 1       1  
Electrical         1  
Bsmt Unf SF        1  
Garage Cars        1  
dtype: int64
```

Удаление пустых значений

Чтобы удалить все строки с пустыми значениями нужно использовать:

```
In [30]: df.dropna()  
pass # чтобы не было вывода результата прошлой ф-ции
```

О нет, постойте, я хотел сказать: "Это нужно **НИКОГДА НЕ ИСПОЛЬЗОВАТЬ!**". Ну, за исключением отдельных случаев, когда потеря сопутствующих данных минимальна или есть другие причины, позволяющие выкинуть часть данных в мусорку.

Как нужно поступать? Ну, вариантов достаточно много, но мы рассмотрим самые распространённые.

Первый вариант --- удалить строки с пустыми значениями в конкретном столбце. Например, мы видим, что в `Garage Area` есть только одно пустое значение. Если мы удалим одну строку из почти что трёх тысяч, то почти наверняка ничего ужасно страшного не случится. Сделаем это.

```
In [31]: df.dropna(subset=['Garage Area'])  
pass
```

Проверяем

```
In [32]: df.isna().sum().sort_values(ascending=False)[lambda x: x > 0][-1:]
```

```
Out[32]: Garage Cars      1  
          dtype: int64
```

Ой, ничего не удалилось, как же так?

`dropna` не изменяет оригинальный датафрейм, а возвращает новый. Выхода два: переприсваивать или менять на месте (да, правильно, во многих функциях выше была такая же ситуация).

```
In [33]: df = df.dropna(subset=['Garage Area']) # переприсваивание  
df.dropna(subset=['BsmtFin SF 2'], inplace=True) # изменить в датафрейме
```

Заполнение пропусков

Рассмотрим самые простые способы заполнения пропусков. Их больше, но они выходят за рамки нашего курса. Самое главное --- нужно понимать какой метод к чему применять и зачем это делать. По идее, это должны рассказать в теории, поэтому кратко:

- 0, `No`, `False` и прочие вещи, указывающие на наличие отсутствия: если не указано, то значит этого нет;
- Среднее;
- Медиана;
- Мода (иногда используется для категориальных переменных; надеюсь вам рассказывали про типы фич);

```
In [34]: df['Lot Frontage'].fillna(0, inplace=True) # заполняем нулями  
df['Mas Vnr Area'].fillna(df['Mas Vnr Area'].mean(), \  
                           inplace=True) # среднее, для медианы нужна  
                           # mean заменить на median  
df['Garage Cond'].fillna(df['Garage Cond'].mode()[0], \  
                           inplace=True) # мода, возвращает  
                           # последовательность,  
                           # если мод несколько
```

```
C:\Users\ShkuratD\AppData\Local\Temp\ipykernel_13388\3652531475.py:1: FutureWarning:  
  A value is trying to be set on a copy of a DataFrame or Series through chaine  
d assignment using an inplace method.
```

The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.

```
df['Lot Frontage'].fillna(0, inplace=True) # заполняем нулями  
C:\Users\ShkuratD\AppData\Local\Temp\ipykernel_13388\3652531475.py:2: FutureWarning:  
  A value is trying to be set on a copy of a DataFrame or Series through chaine  
d assignment using an inplace method.
```

The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.

```
df['Mas Vnr Area'].fillna(df['Mas Vnr Area'].mean(), \  
C:\Users\ShkuratD\AppData\Local\Temp\ipykernel_13388\3652531475.py:5: FutureWarning:  
  A value is trying to be set on a copy of a DataFrame or Series through chaine  
d assignment using an inplace method.
```

The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.

```
df['Garage Cond'].fillna(df['Garage Cond'].mode()[0], \
```

Для любознательных: узнать, что изменится в pandas 3.0 и как устраниТЬ (не отключить) предупреждение в коде выше.

Проверка на дубликаты

```
In [35]: df.duplicated().sum()
```

```
Out[35]: np.int64(0)
```

Дубликатов нет, но если бы они были, мы могли бы их убрать с помощью df.drop_duplicates().

Фильтрация

Мы можем использовать булеву индексацию для фильтрации датафрейма.

```
In [36]: df_filtrated = df[df['Mas Vnr Area'] > 10]
len(df_filtrated)
```

```
Out[36]: 1178
```

Фильтрация создаёт копию, при попытке изменить что-либо в отфильтрованном датафрейме появляется предупреждение, что оригинальный датафрейм не изменится и что следует пользоваться индексацией.

```
In [37]: df_filtrated['Mas Vnr Area'] = 0
```

```
C:\Users\ShkuratD\AppData\Local\Temp\ipykernel_13388\2247710711.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
df_filtrated['Mas Vnr Area'] = 0
```

Индексация

Есть два способа индексации датафреймов: `loc` и `iloc`. Первый проводит индексацию по меткам, второй по индексам. Ниже выберем вторую строку (`Order=2`), столбец `Mas Vnr Area`. Осторожно, если мы удалили строку с таким индексом, то получим ошибку.

```
In [38]: df.loc[2, 'Mas Vnr Area']
```

```
Out[38]: np.float64(0.0)
```

Также можно выбирать несколько значений. Два столбца, строки с первой по третью *включительно*.

```
In [39]: df.loc[1:3, ['Lot Frontage', 'Mas Vnr Area']]
```

```
Out[39]:    Lot Frontage  Mas Vnr Area
```

Order		
1	141.0	112.0
2	80.0	0.0
3	81.0	108.0

Для `iloc` всё тривиально.

```
In [40]: df.iloc[9, 6]
```

```
Out[40]: 'Reg'
```

```
In [41]: df.iloc[:5, :6]
```

```
Out[41]:
```

	MS Zoning	MS SubClass	Lot Frontage	Lot Area	Street	Alley
Order						
1	RL	20	141.0	31770	Pave	NaN
2	RH	20	80.0	11622	Pave	NaN
3	RL	20	81.0	14267	Pave	NaN
4	RL	20	93.0	11160	Pave	NaN
5	RL	60	74.0	13830	Pave	NaN

Для любознательных: узнать зачем нужен `at[]` и `iat[]`.

Обратим внимание, что максимальный индекс нашего датафрейма:

```
In [42]: print(f'{df.index.max()}', '# запятая просто для красоты')
```

2930,

а данных (строк) у нас всего:

```
In [43]: print(f'{len(df)}', '# точка просто для красоты')
```

2928.

Вы же помните, что мы ранее удалили две строки? Так вот индекс не пересчитался.

В таких случаях полезно восстановить индексацию (да и не только в таких, при разделении одного датафрейма на несколько тоже помогает).

```
In [44]: df.reset_index().head()
```

	Order	MS Zoning	MS SubClass	Lot Frontage	Lot Area	Street	Alley	Lot Shape	Land Contour	Utilities
0	1	RL	20	141.0	31770	Pave	NaN	IR1	Lvl	AllPub
1	2	RH	20	80.0	11622	Pave	NaN	Reg	Lvl	AllPub
2	3	RL	20	81.0	14267	Pave	NaN	IR1	Lvl	AllPub
3	4	RL	20	93.0	11160	Pave	NaN	Reg	Lvl	AllPub
4	5	RL	60	74.0	13830	Pave	NaN	IR1	Lvl	AllPub

5 rows × 81 columns

Вроде бы всё хорошо, но если мы посмотрим на датафрейм, то увидим, что наш столбец `Order` снова стал признаком, чего нам не надо. Для этого используем параметр `drop`. Обратите внимание, что в строке выше не использовалось присваивание и аргумент `inplace`.

```
In [47]: df.reset_index(drop=True, inplace=True)  
df.head()
```

Out[47]:

	MS Zoning	MS SubClass	Lot Frontage	Lot Area	Street	Alley	Lot Shape	Land Contour	Utilities	Lot Config
0	RL	20	141.0	31770	Pave	NaN	IR1	Lvl	AllPub	Corner
1	RH	20	80.0	11622	Pave	NaN	Reg	Lvl	AllPub	Inside
2	RL	20	81.0	14267	Pave	NaN	IR1	Lvl	AllPub	Corner
3	RL	20	93.0	11160	Pave	NaN	Reg	Lvl	AllPub	Corner
4	RL	60	74.0	13830	Pave	NaN	IR1	Lvl	AllPub	Inside

5 rows × 80 columns

Группировка и агрегирование

Мы можем сгруппировать данные по одному или нескольким признакам, а потом задать функцию, которая выдаст информацию о желаемом признаке. На практике понятнее. Допустим, мы хотим узнать среднюю стоимость домов по годам.

In [52]: `df.groupby('Yr Sold')['SalePrice'].mean()`

Out[52]: `Yr Sold`

2006	181761.648000
2007	185187.600289
2008	179002.526570
2009	181404.567901
2010	172597.598240

Name: SalePrice, dtype: float64

По тому же принципу можем узнать общий объём продаж (`sum`), количество (`count`), минимальную и максимальную сумму (`min` и `max` соответственно), медиану (`median`) и другое.

Агрегация же позволяет применить несколько агрегатных функций (как те, что выше) одновременно. Для этого нужно передать словарь в функцию `agg`.

In [54]: `df.groupby('Yr Sold').agg({`

```
'SalePrice': ['min', 'median', 'mean', 'max'],
'Lot Area': ['mean', 'std']
})
```

Out[54]:

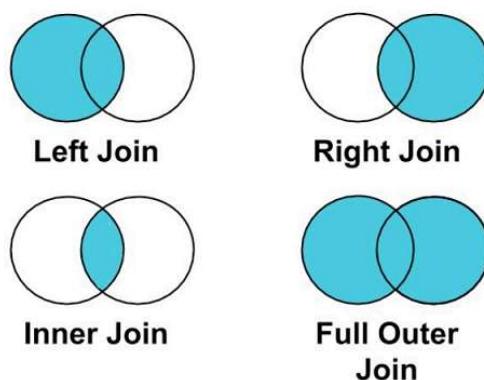
Yr Sold	SalePrice				Lot Area	
	min	median	mean	max	mean	std
2006	35000	159500.0	181761.648000	625000	10205.915200	5835.055266
2007	39300	165250.0	185187.600289	755000	10472.109668	8740.105014
2008	13100	161000.0	179002.526570	615000	10164.041868	8504.649454
2009	34900	160850.0	181404.567901	610000	9953.790123	9052.532833
2010	12789	155000.0	172597.598240	611657	9737.876833	5348.890282

Слияние и конкатенация

Слияние

Слияние работает по тем же принципам, что и в SQL.

SQL Joins



Возьмём несколько столбцов и объединим их по индексу.

Out[59]:

	Neighborhood	House Style	Year Built	Bldg Type	SalePrice	Gr Liv Area	Lot Area
0	NAmes	1Story	1960	1Fam	215000	1656	31770
1	NAmes	1Story	1961	1Fam	105000	896	11622
2	NAmes	1Story	1958	1Fam	172000	1329	14267
3	NAmes	1Story	1968	1Fam	244000	2110	11160
4	Gilbert	2Story	1997	1Fam	189900	1629	13830

Вообще если бы мы оставили `Order` (изначальное индексирование) или `PID` (уникальный ключ), то мы бы могли провести объединение по ним, передав как аргумент в `on`, но мы от того и от другого избавились, так что можете попробовать сделать это сами. Не обязательно даже датасет перезагружать, можете просто создать новый столбец, который будет индексом (не в смысле индексом имеющегося датафрейма, т.е. чтобы мы не могли его получить через `df.index`, а просто столбцом). Кстати:

Для любознательных: сделать выше описанное оптимальным образом. Подсказка: посмотрите на тип индекса датафрейма.

Конкатенация

Это просто объединение двух датафреймов в один. Есть

Вертикальная конкатенация

Это когда один датафрейм подставляем под (ниже) другой, изменяется количество строк, но не столбцов. Для примера разделим наш датасет на две части и соединим снова.

```
In [61]: n = len(df)

df_part1 = df.iloc[:n//2] # деление нацело, последний не включается
df_part2 = df.iloc[n//2:]

df_concat = pd.concat([df_part1, df_part2], axis=0) # 0 -- вертикальная,
# добавляем строки
print(df_concat.shape)
```

(2928, 80)

Ещё есть

Горизонтальная конкатенация

Можно было бы подумать, что в горизонтальной конкатенации случай обратный, добавляются только столбцы, но не строки, но не всё так просто. При горизонтальной конкатенации строки выравниваются по индексам. Если в одном датафрейме есть строка с индексом N, а в другом нет, то в строке с индексом N объединённого датафрейма (не теряемся) столбцы второго датафрейма будут NaN, поскольку данных для его объединения не было, т.е. количество строк изменится.

Можете проверить это сами, я продемонстрирую более скучный вариант конкатенации.

```
In [62]: pd.concat([df_info, df_price], axis=1).head() # 1 -- горизонтальная
```

Out[62]:

	Neighborhood	House Style	Year Built	Bldg Type	SalePrice	Gr Liv Area	Lot Area
0	NAmes	1Story	1960	1Fam	215000	1656	31770
1	NAmes	1Story	1961	1Fam	105000	896	11622
2	NAmes	1Story	1958	1Fam	172000	1329	14267
3	NAmes	1Story	1968	1Fam	244000	2110	11160
4	Gilbert	2Story	1997	1Fam	189900	1629	13830

Важно: слияние и горизонтальная конкатенация --- это разные вещи. Первое более гибкое, позволяет выбрать способ слияния, можно использовать в качестве ключа слияния не индексы, второе более прямолинейное, создающее пустые значения, объединять можно только по индексам.

Для любознательных

(напоминание, ссылки почему-то работают только в html и pdf версии, VS Code не хочет)

1. [Здесь](#)
 2. [Здесь](#)
 3. [Здесь](#)
-

P.S. Если у нас кастомные индексы, то они не продолжатся автоматически. Убедитесь, что новая добавляемая последовательность не нарушает логику индексации старой (или убедитесь, что вас это не волнует в достаточной мере). Ну или используйте `reset_index`.

```
In [46]: series = pd.Series([10, 20, 30], index=['a', 'b', 'c'])
series = pd.concat([series, pd.Series([40])])
series
```

Out[46]:

```
a    10
b    20
c    30
0    40
dtype: int64
```