* **Библиотека pandas**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

import statsmodels.api as sm

import statsmodels.formula.api as smf

from statsmodels.stats.api import anova\_lm

from statsmodels.stats.multicomp import (pairwise\_tukeyhsd,

MultiComparison)

from scipy import stats

import pingouin as pg

import numpy as np

import scipy.stats as st

import pandas as pd

from sklearn import metrics

from sklearn.linear\_model import LinearRegression

* **Чтение данных в формате CSV**

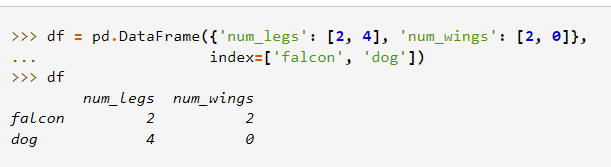
df = pd.read\_csv('2\_bookings.csv', sep = ';', parse\_dates = ['arrival full date'])

df = pd.read\_excel('dataset\_pasta (06.09).xlsx', sheet\_name='Лист1')

* **Создание датафрейма**
* создаем датафрейм

df = pd.DataFrame(np.random.randn(7, 2))

df



* df = pd.DataFrame(np.random.randn(100, 3), index=pd.date\_range('1/1/2022', periods=100),

columns=['A', 'B', 'C'])

* right = pd.DataFrame({'name': ['Ivan', 'Peter', 'Elena', 'Sveta', 'Irina', 'Ivan'], 'rval': [6, 5, 7, 9, 10, 6]})



* df\_data = pd.DataFrame([['Anna', 23, 3],

['Ivan', 36, 12],

['Lena', 33, 10],

['Irina', 25, 7],

['Lisa', 27, 7],

['Peter', 32, None]])

df\_data.columns = ['name', 'age', 'expr']

df\_data

* **Разбитие датафрейма на части**

**#разбиваем на части датафрейм**

**parts = [df[:2], df[2:5], df[5:]] #разбиваем на части**

* **Характеристики датафрейма**

#получение информации о датафрейме

df.info()

count\_values – это ты? :

data\_adj.groupby('cluster').size()

#подсчет количества строк и столбцов

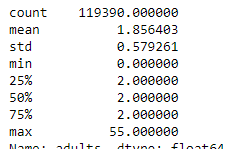
df.shape

#типы данных

df.dtypes

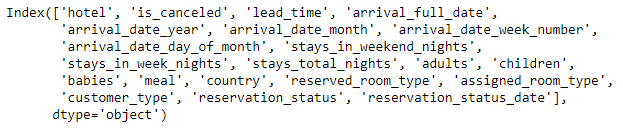
#расчет описательных

df.describe()



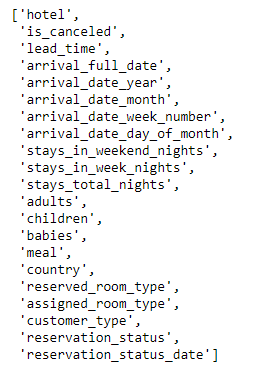
#вывод всех колонок

df.columns



Чтобы получить список названий, достаточно сконвертировать тип с помощью привычного **list()**

list(df.columns)



***#создание функции***

***def first\_check(dataset):***

***print()***

***print('Первые 3 строки таблицы')***

***display(dataset.head(3))***

***print()***

***print('Последние 3 строки таблицы')***

***display(dataset.tail(3))***

***print()***

***print('Информация о таблице')***

***print(dataset.info())***

***print()***

***print('Характеристики чисел в таблице')***

***display(dataset.describe())***

***print()***

***print('Наименование колонок')***

***print(dataset.columns)***

***print()***

***print('Типы колонок')***

***print(df.dtypes)***

***print()***

***print('Типы колонок')***

***print(df.shape)***

* **Переименование столбцов**

#переименование столбцов

df = df.rename(columns={'reserved\_room\_type': 'reserved\_room', 'assigned\_room\_type': 'assigned\_room'})

df = df.rename(columns=lambda c: c.lower().replace('-', '\_').replace('/', '\_').replace(' ', '\_'))

* **Удаление пропущенных значений - dropna**

df\_data = df\_data.**dropna**()

df\_data

* **Обращение к колонкам**

df.column\_name

#Что делать, если название колонки состоит из 2-х слов?

#Либо переименовать колонку, либо использовать другой способ доступа:

df['column name']

df[['column1', 'column2', 'column3']]

* **Выбор строк по условиям**

df\_data[df\_data['age'] pivor30]

несколько условий:

1. Два условия

# если без первого df\_data[...], то выведет True/False

df\_data[(df\_data['age'] > 30) & (df\_data['expr'] > 7)] # не забываем круглые скобки

1. Или одно, или другое:

df\_data[(df\_data['age'] > 35) | (df\_data['age'] < 25)]

* **Создание столбцов**

1. Возведение в степень

df\_data['age\_sq'] = df\_data['age'] \*\* 2

1. Вычитание

df\_data['no\_work'] = df\_data['age'] - df\_data['expr']

1. Из одного значения

df\_data['status'] = 'W' # из одного значения

1. Из списка значений

df\_data['gender'] = [0, 1, 1, 0, 0]

1. создание столбца True/False по условию

df\_data['diff'] = df\_data.age>27

* **Вычисление**

df.money.sum() - аналогично другие аггрегирующие функции (min, max, mean, var(расчет дисперсии), std(расчет среднеквадратического отклонения)

Для выбора строк и столбцов в Pandas есть два основных метода**: .iloc и .loc**. Первый используется для выбора строк и столбцов по их индексу, второй – по их названию. Внутри каждого из этих методов в квадратных скобках указывается сначала идентификатор (индекс или название) строки, а затем – идентификатор столбца.

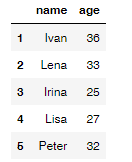
* **Метод loc**

df\_data\_new = df.loc[:, 'Adults':'Babies']

#строки, затем столбцы

* **Метод iloc**

df\_data.iloc[1:7, 0:2]



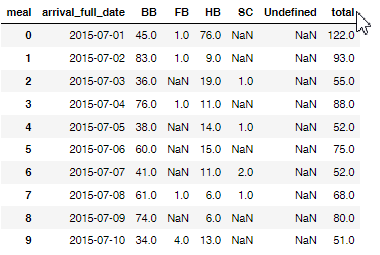
* **Добавить индекс:**

**df\_data\_new.index** = df\_data\_new.name

df\_data\_new

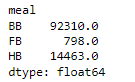
* **Метод apply**

df\_pivot['total'] = df\_pivot.loc[:,'BB':'HB'].apply(np.sum, axis=1)



Если бы мы написали **axis=0,** то получили бы cумму броней по выбранным типам питания (три значения):

df\_pivot.loc[:,'BB':'HB'].apply(np.sum, axis=0)



* **Функция lambda:**

Самое интересное и полезное: в .apply() можно прописывать свои функции, которые мы заранее определим. Напишем функцию, которая будет считать размах: вычитать из максимального значения минимальное и возвращать результат. Напишем небольшую lambda-функцию в Python и назовём её f.

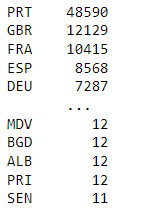
Сначала мы указываем название функции, потом после знака равенства начинаем её определять. Стартуем с ключевого слова lambda, чтобы Python понимал, что это функция. После указываем аргумент – то, с чем функция должна работать, то, что подаётся ей на вход. Назвать его мы можем как угодно, у нас будет x. Далее через двоеточие мы прописываем, что с этим x нужно сделать, то есть вернуть на выходе, в результате исполнения функции.

**f = lambda x: x.max() - x.min()**

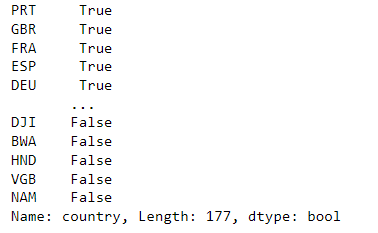
* **Функция lambda+loc/lambda + apply:**

1. **Lambda+loc**

**df['country'].value\_counts().loc[lambda x : x > 10]**



1. **df['country'].value\_counts().apply(lambda x : x > 10)**



**df\_pivot['diff'] = df\_pivot.loc[:,'BB':'SC'].apply(f, axis=1)**

* **Value\_counts()**

Функция value\_counts() используется для получения Series , содержащего уникальные значения. Она вернет результат, отсортированный в порядке убывания, так что первый элемент в коллекции будет самым встречаемым.

NA-значения не включены в результат

Важно заметить, что value\_counts работает только с series, но не dataframe. Поэтому нужно указать одни квадратные скобки df['your\_column'], а не пару df[['your\_column']].

**Параметры:**

* normalize (по умолчанию False) — если True, то в процентах.
* sort (по умолчанию True) — сортировка по частоте.
* ascending (по умолчанию False) — сортировка по убыванию.
* bins (int) — вместе подсчета значений группирует их по отрезкам, но это работает только с числовыми данными.
* dropna (по умолчанию True) — не включать количество NaN.

**df['country'].value\_counts() #или df.country.value\_counts()**

* **из серии в ДатаФрейм (DataFrame)**
* df['country'].value\_counts().to\_frame()
* value\_counts = df['country'].value\_counts()

# преобразование в df и присвоение новых имен колонкам

df\_value\_counts = pd.DataFrame(value\_counts)

df\_value\_counts = df\_value\_counts.reset\_index()

df\_value\_counts.columns = ['country', 'count']

* **Агрегирование - Groupby**

Groupby — очень популярный метод в Pandas. С его помощью можно сгруппировать результат по одной колонке и посчитать значения в другой.

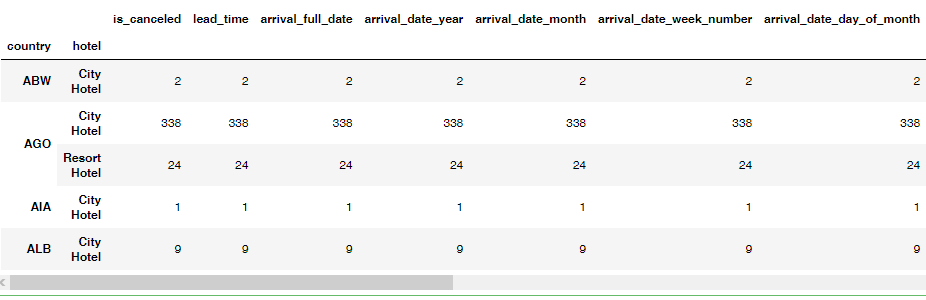
* df.groupby('your\_column\_1')['your\_column\_2'].value\_counts()

df.groupby('hotel')['country'].value\_counts()

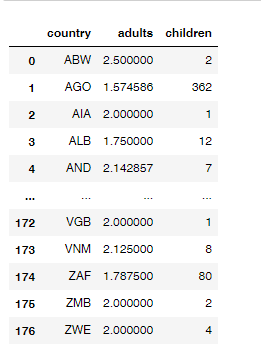
Это мульти-индекс, позволяющий иметь несколько уровней индексов в dataframe. В этом случае тип отеля соответствует нулевому уровню индекса, а страна — первому.

**df.groupby('hotel')['country'].value\_counts()**

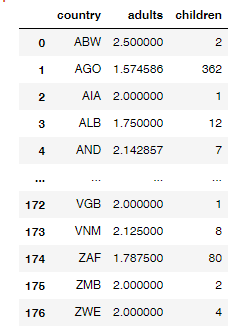
* df.groupby(['country', 'hotel']).count().head(5)



* df.groupby(["country"]).agg({"adults": "mean", "children": "count"}).reset\_index()



* df.groupby(["country"]).agg({"adults": "mean", "children": "count"}).reset\_index()



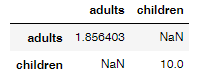
* df['adults'].agg(['sum', 'mean']) #или для обращения к различным столбцам



* #Использование кортежей (именованное агрегирование)

df.agg(adults=('adults', 'mean'),

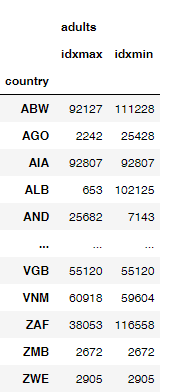
children=('children', 'max'))



* Использование idxmax и idxmin для выбора значения индекса, соответствующего максимальному или минимальному значениям.

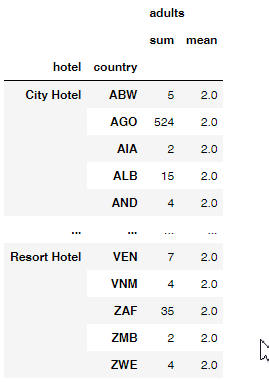
agg\_func\_max\_min = {'adults': ['idxmax', 'idxmin']}

df.groupby(['country']).agg(agg\_func\_max\_min)



* агрегирование по столбцам и по 2 агрегирующим значениям

df.groupby(['hotel', 'country']).agg({'adults': ['sum', 'mean']}).round()



* **Удаление заданных столбцов**

df\_data.drop(['expr'], axis=1).head()

* **Конкатенация двух датафреймов**

Эта методика применима в ситуациях, когда имеются два датафрейма **с одинаковыми столбцами**, которые нужно скомбинировать.

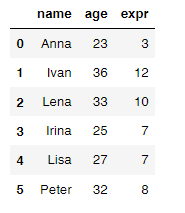
#разделим предыдущий датафрейм

df1 = df\_data[0:2]

df2 = df\_data[2:6]

#объединим два датафрейма

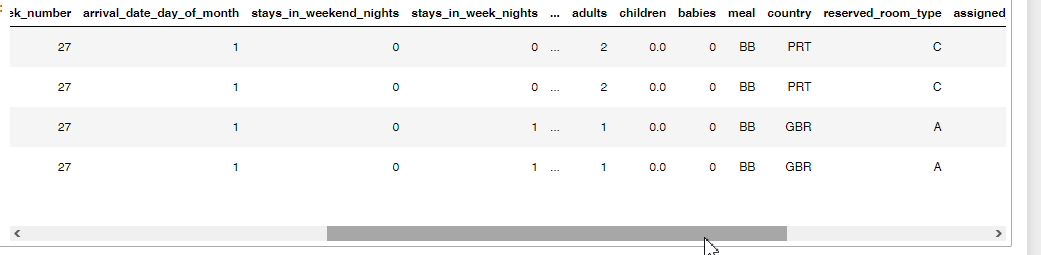
**pd.concat([df1, df2],** ignore\_index=True)



* **Получение строк по заданным значениям столбцов**

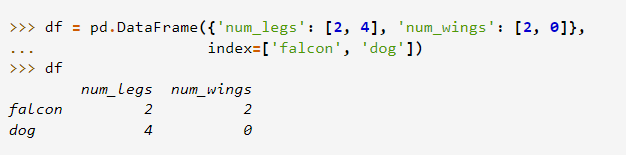
1. Для получения строк датафрейма в ситуации, когда имеется список значений столбцов, можно воспользоваться следующей командой **isin:**

**df[df['country'].isin(['PRT', 'GBR'])].head(4)**

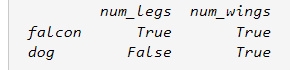


**Isin** – можно сделать проверку, входят ли значения в данный датафрейм:

У нас есть датафрейм:



**df.isin([0, 2])**



1. Если нас интересует единственное значение — можно воспользоваться такой конструкцией:

**df[df['country'] == 'PRT'].head(4)**

1. Получение среза датафрейма

**df[1:5]**

1. Фильтрация по значению

**df[df['adults'] > 4].head()**

* **Сортировка**
* Для сортировки датафреймов по значениям столбцов можно воспользоваться функцией sort\_values()

**df.sort\_values('adults', ascending=False).head(10) #ascending=False - по убыванию**

* **Метод query**

В пандасе есть возможность фильтровать данные используя SQL-like синтаксис. Для этого нужен метод query, принимающий строку с запросом. Внутри него можно использовать названия колонок (если они без пробелов).

При использовании строк внутри запроса экранируйте кавычки \ или используйте другую пару

В query также можно передать сразу несколько условий. Условия, которые должны выполняться одновременно, соединяются с помощью and или &

Когда должно удовлетворяться одно из условий – or или |:

df.query("country == 'ABW' and hotel == 'City Hotel'")

* **Функция replace**

replace – применяется к строкам, принимает 2 строки: что заменить и на что

my\_message = 'Hello world'

my\_message.replace('o', '\_')



* **Функция strip**

применяется к строкам, по умолчанию убирает пробелы слева и справа

my\_message = ' Hello world '

my\_message.strip()

**Время**

Для работы с датой и временем можно использовать модуль datetime. Для получения данных о времени в момент вызова функции используйте функцию **today** в одноимённом подмодуле.

* **import datetime**

# by convention imports are placed in the head of file and separated with 2 blank lines from other code

date = datetime.datetime.today()

date



* **today**

**Перевести в строку datetime:**

date = datetime.datetime.today().**strftime**('%Y-%m-%D-%H:%M:%S')

date

*strftime форматирует дату по переданному ему формату:*

*% – обозначает что дальше будет часть даты Y – год 4-мя знаками m – месяц 2-мя знаками d – день H – час M – минуты S – секунды*



* **now**

from datetime import datetime

# current date and time

now = datetime.now()

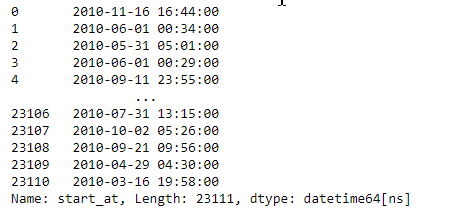
print(f'Full time format of now is {now}')

* **Конвертация во время**

**pd.to\_datetime()** — метод, позволяющий превратить строки во время.

Это позволяет удобно с ним работать.

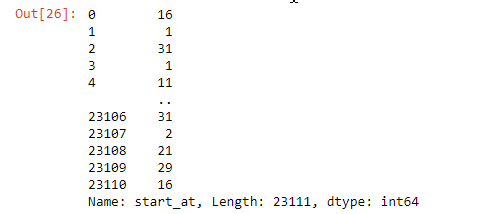
df['Date'] = pd.to\_datetime(df['Date'])



df['start\_at'] = pd.to\_datetime(df.start\_at)

* **Day**

df.start\_at.dt.day



* weekday

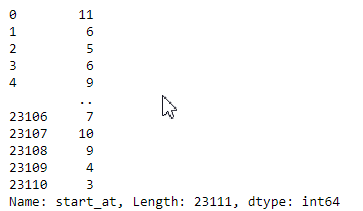
df.start\_at.dt.weekday

* Day\_name

df.start\_at.dt.day\_name()

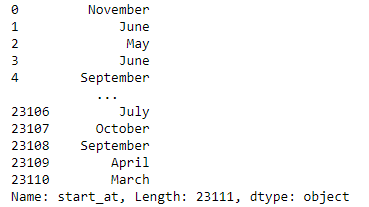
* month

df.start\_at.dt.month



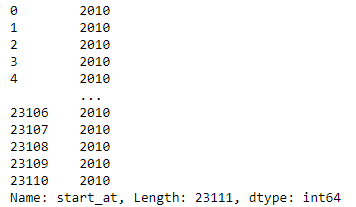
* month\_name

df.start\_at.dt.month\_name()

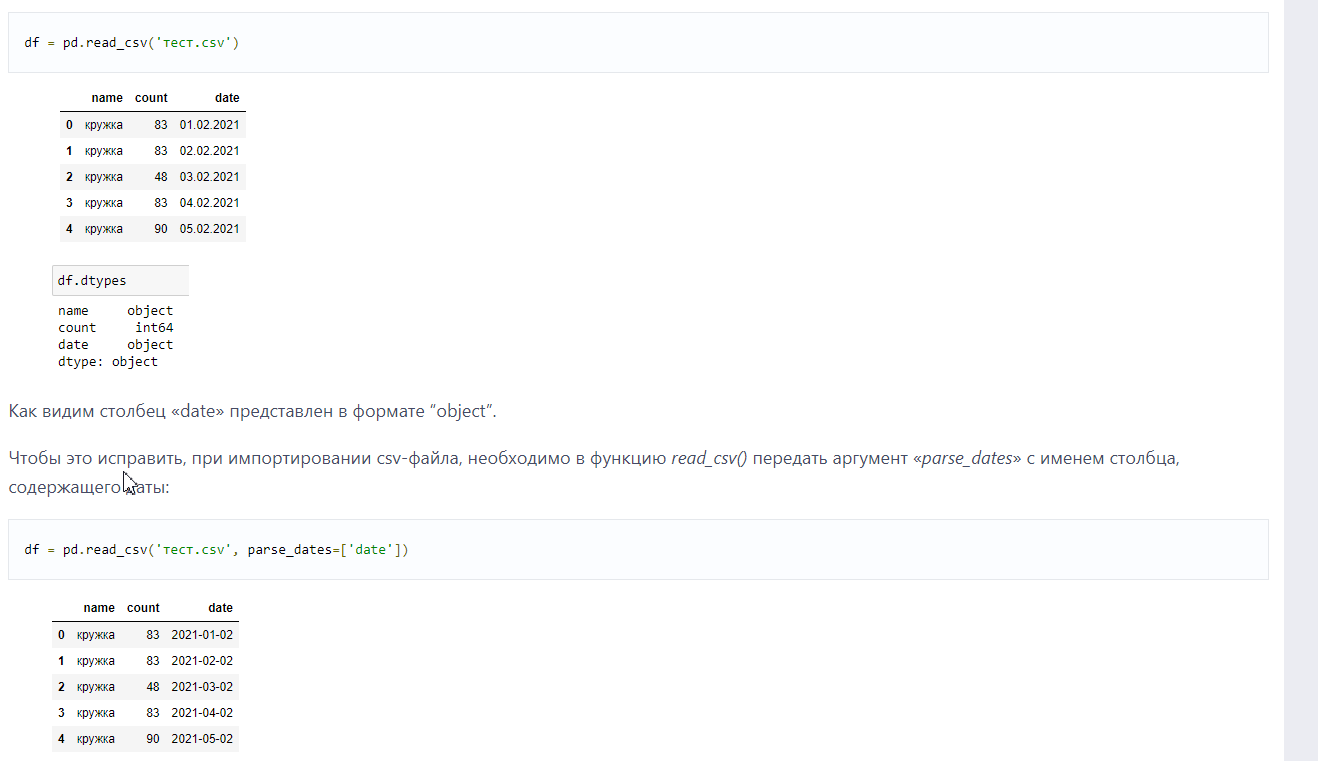


* year

df.start\_at.dt.year



* parse\_dates

df = pd.read\_csv('тест.csv', parse\_dates=['date'])

* Как извлечь из даты информацию о месяце и годе?

создание столбца year\_month

df\_orders['year\_month'] = df\_orders['order\_date'].dt.to\_period('M')

* интервал между двумя датами

df\_orders['duration\_order'] = df\_orders.ship\_date - df\_orders.order\_date

df\_orders.head(2)



df\_orders['days\_to\_order'] = (df\_orders['ship\_date'] - df\_orders['order\_date']).dt.days

df\_orders.head(2)

* Прибавление интервала

df\_orders['plan\_date'] = df\_orders.order\_date+pd.DateOffset(days=2)

df\_orders.head(2)

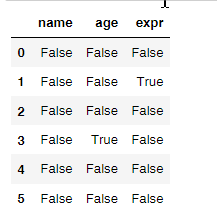


* **Поиск пустых значений – isna**

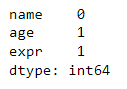
**isna — это метод, с помощью которого можно быстро найти пропущенные значения в датафрейме:**

**Применив его, на выходе мы получаем датафрэйм той же размерности, где в каждой ячейке True или False — в зависимости от того, было ли значение пропущено:**

* df\_data.isna()



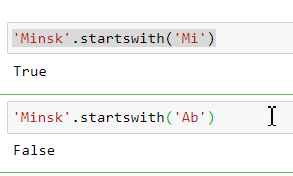
* df\_data.isna().sum()



* Проверка на начало строки - startswith

startswith — строковый метод, принимающий другую строку и возвращающий True или False в зависимости от того, начинается ли исходная строка с переданной.

**'Minsk'.startswith('Mi')**



my\_str = 'Discworld'

my\_str.startswith('Mad') # False

my\_str.startswith('Disc') # True

my\_str.startswith(('Disc', 'Mad')) # True

my\_str.startswith('world', 4, 9) # True

* Изменить тип датафрейма – astype

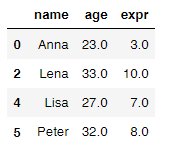
df = df.astype({'money': 'float'}) # df.money will be rational number after this line

df.height = df.height.astype('float') # df.height will be rational number

* Удаление пропущенных значений – dropna

dropna — метод, позволяющий выкинуть из датафрэйма все строки, содержащие пропущенные значения.

df\_data.dropna()



**Merge/Concat**

* **Concat**

# 

# parts = [df[:2], df[2:5], df[5:]]

# 

# pd.concat(parts)

* **Merge**

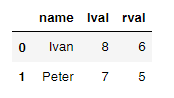
# Merge зачастую называется джойном. Очень частая операция, которую можно сделать с помощью нескольких функций. Одна из них — merge. Обязательным аргументом является другой датафрейм, с которым планируется объединение. Объединение идёт по общей колонке, у которой имеется одинаковый смысл и общие значения в обоих датафреймах.

# разница между объеднинениями проиллюстрирована ниже

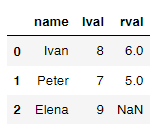
# Изначальные данные:

# 

* pd.merge(left, right, on='name') *#здесь идет how = 'inner'*



* pd.merge(left, right, on='name', how = 'left')



* pd.merge(left, right, on='name', how = 'right')
* **Join**

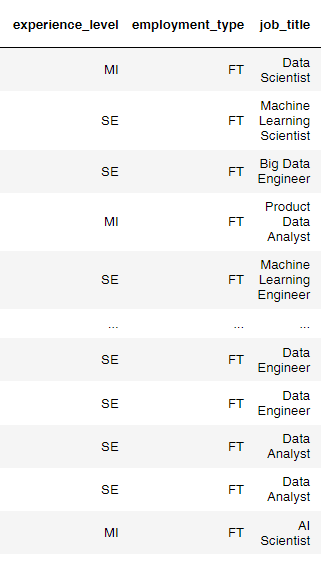
Метод join в Python отвечает за объединение списка строк с помощью определенного указателя

**DataFrame.join(other, on=None, how='left', lsuffix='', rsuffix='', sort=False, validate=None) - https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.join.html**

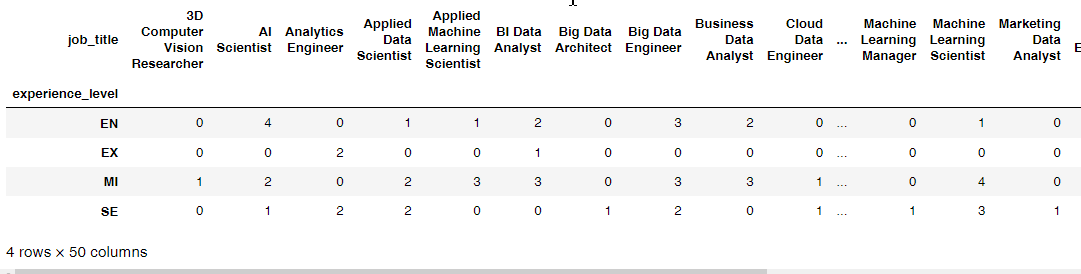
DataFrame, Series, or a list containing any combination of them

* **Сrosstab**

Функция crosstab создает таблицу кросс-табуляции, которая может показать частоту, с которой появляются определенные группы данных.



* **pd.crosstab(df.experience\_level, df.job\_title)**



* В кросс-таблице возможно включение промежуточных итогов с помощью ключевого слова margins:

pd.crosstab(df.experience\_level,

df.company\_size,

margins=True,

margins\_name="Total")

* crosstab позволяет указывать значения для агрегирования в aggfunc='mean' (max, min, sum)

pd.crosstab(df.experience\_level,

df.company\_size,

values=df.salary\_in\_usd,

aggfunc='mean').round(2)

* C помощью параметра normalize мы можем понять, сколько процентов от общего числа составляет каждая комбинация:

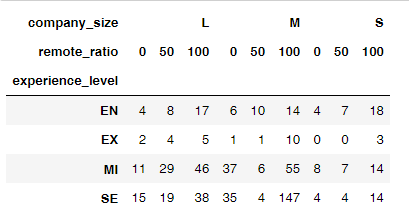
pd.crosstab(df.experience\_level,

df.company\_size,

normalize=True)

* Можно для группировок дополнительно указывать несколько столбцов в виде списка:

pd.crosstab(df.experience\_level, [df.company\_size, df.remote\_ratio])

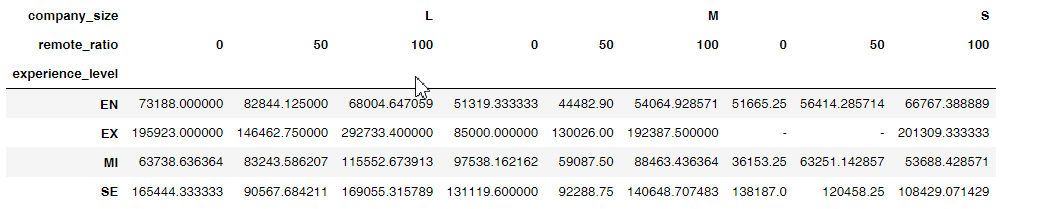


* Можете использовать функции агрегирования при группирвке:

pd.crosstab(df.experience\_level, [df.company\_size, df.remote\_ratio],

values=df.salary\_in\_usd,

aggfunc='mean').fillna('-')



1. **Аналогично groupby и unstack**

df.groupby(['experience\_level', 'job\_title'])['job\_title'].count().unstack().fillna(0)

1. **pivot\_table**

df.pivot\_table(index='experience\_level',

columns='job\_title',

aggfunc={'job\_title':len},

fill\_value=0)

# Создание сводной таблицы: pivot\_table

Сводные таблицы — удобный способ преобразовать данные, с возможностью применения к ним агрегирующей функции. В pandas есть 2 функции, различающиеся только тем, проводится ли агрегация.

https://pandas.pydata.org/docs/reference/api/pandas.pivot\_table.html

**Обе принимают 3 аргумента:**

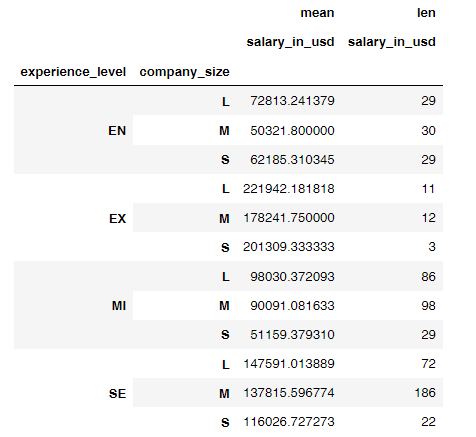
* index — название колонки, значения из которой станут индексами.
* columns — название колонки, значения из которой станут колонками.
* values — название колонки, значения из которой распределятся по сформированным группам.

1. pd.pivot\_table(df,

index=["experience\_level", "company\_size"],

values=["salary\_in\_usd"],

aggfunc=[np.mean, len])



1. pd.pivot\_table(df,

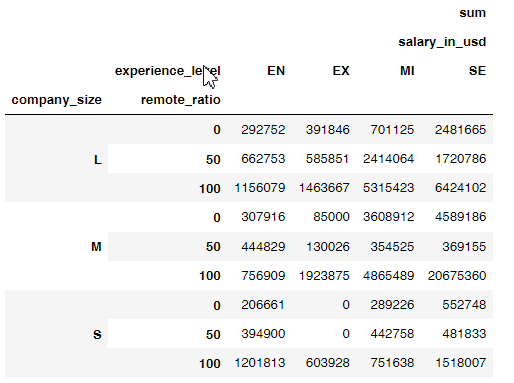
index=['company\_size', 'remote\_ratio'],

values=["salary\_in\_usd"],

columns=["experience\_level"],

aggfunc=[np.sum],

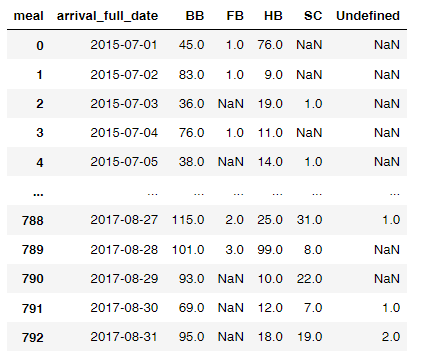
**fill\_value=0)**



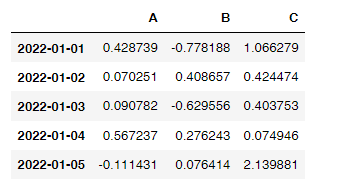
5. Создание сводной таблицы

df\_pivot = df.pivot\_table('adults', index='arrival\_full\_date', columns='meal', aggfunc = 'count').reset\_index()

'adults' – по какой колонке фильтруем

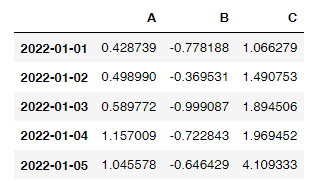


* Cumsum – использование кумулятивной суммы

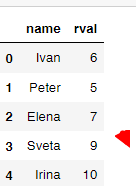


df = df.cumsum()

df.head()

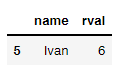


* Фильтрация дубликатов
* df = right.drop\_duplicates()



* просмотр дубликатов

right.loc[right.duplicated()]



1. просмотр определённых колонок с дубликатами

df.drop\_duplicates(subset=['name\_col1', 'name\_col2'])

**Очищаем значения:**

1. **dropna** - удаление пропущенных значений

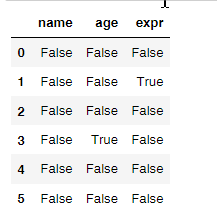
df\_data = df\_data.**dropna**()

1. **nunique - количество уникальных**

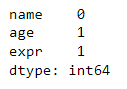
df.query('date == "2020-01-01"').device\_id.**nunique()**

1. **Isna - поиск пустых значений**

* df\_data.isna()



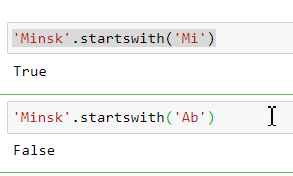
* df\_data.isna().sum()



1. **Проверка на начало строки - startswith**

startswith — строковый метод, принимающий другую строку и возвращающий True или False в зависимости от того, начинается ли исходная строка с переданной.

**'Minsk'.startswith('Mi')**



my\_str = 'Discworld'

1. Fillna – заменяет пропуск NaN на другое значение

pd.crosstab(df.experience\_level, [df.company\_size, df.remote\_ratio],

values=df.salary\_in\_usd,

aggfunc='mean').**fillna('-')**

1. Выборка определённых значений

df\_1 = df\_1.query('Quantity<10000') or df[df['country'] == 'PRT'].head(4)