**Numpy and Pandas**

import pandas as pd

import numpy as np

|  |  |
| --- | --- |
| **Numpy** |  |
| a = np.array([ [1,2,3],[4,5,6] ]) | Создание списка NumPy |
| a.ndim >>> 2 | **количество измерений** массива (линейный, матрица, 3-мерный и т.д.) |
| a.size >>> 6 | **количество элементов** в массиве |
| a.shape >>> (2,3) | **форма** списка (2 строки по 3 элемента) |
| a.dtype >>> ‘int32’ | **Формат** **данных** в списке |
| a.sum() >>> 21  a.prod() >>> 720  a.sum(axis=0) >>> array([5,7,9])  a.sum(axis=1) >>> array([6,15]) | сумма всех элементов  произведение всех элементов  сумма всех строчек поэлементно  сумма значений в каждой строчке |
| a.arange(15) | создать одномерный массив из range(15) - от 0 до 14, по умолчанию шаг =1 |
| a.arange(15).reshape(3,5) | **переформировать** массив из 15 элементов в матрицу с 3 строками по 5 элементов |
| a.reshape(a.shape[::-1]) | **Транспонирование массива** |
| a.arange(0, 4, 0.5)  a.arange(2.3, 5.7, 0.3) | создать одномерный массив - от 0 до 4 с шагом 0.5, по умолчанию шаг =1,  то же самое для дробных значений |
| a=np.linspace(11, 60, 25) | сделать массив float-значений? конечное число - включительно, **с автоматически рассчитываемым шагом** при заданном количестве значений |
| def f(x,y):  return 10\*x+y  np.fromfunction(f, SHAPE, dtype=int) | создание массива из функции |
| a[0,3] >>> 3  a[1, -1] >>> 6 | Выбор значения ячейки по индексам строк, столбцов… |
| a [3:9:2]  a [::-1]  a [1, ::2] >>> [4,6]  a [..., 1] >>> [2,5] | **Срезы**: с 3 по 9 не включительно с шагом 2,  от начала до конца в обратную сторону (развернутый),  из второго элемента срез от начала до конца с шагом 2,  выбор в каждом подмассиве (через троеточие, называемое **Ellipsis**) элемента под индексом 1 |
|  |  |
| size = 10  SHAPE = (2,5)  z = np.zeros(( size, )).reshape(SHAPE)  z = np.zeros(( size, ), dtype=np.int\_).reshape(SHAPE) | создать массив float нулей количеством с заданное число size с заданной формой SHAPE  или то же самое с заданным типом integer |
| z.shape = 5, **-1** | придать массиву форму с автоматическим значением одного из измерений - нужно указать -1 на месте неизвестного |
| с = np.zeros\_like(a) | сделать массив с формой, схожей на форму другого массива (в данном случае массива а) |
|  |  |
| **a.flat**  for elem in a.flat: | генератор по ВСЕМ элементам многомерного массива c перебором всех уровней (можно итерироваться) |
| a.ravel() | перевод значений массива в одномерный массив |
| np.vstack((b,c)) | массив, вертикально (по второму индексу) соединяющий два других массива в единый, сохраняя количество измерений |
| np.hstack((b,c))  либо  np.column\_stack((b,c)) | массив, горизонтально (по первому индексу) соединяющий два других массива в единый, сохраняя количество измерений |
| np.hsplit(a, 2)  np.vsplit(a, 5) | разделить на 2 массива горизонтально  или вертикально |
|  |  |
| np.all(a==b) | **Проверка равенства** всех элементов массивов a и b |
| a + b, a - b, a \* b, a / b, a \* 2, a\*\*4 | поэлементное сложение, вычитание, умножение, деление данных в массивах a и b; поэлементное умножение на 6, возведение в 4 степень |
| np.diff(a) | сделать массив из **разниц между соседними значениями** переданного массива |
| b - a \* c | вычитание из значений массива и произведения значений массивов a и c |
| с = a > 5 | сделать **массив-маску из булевых значений True-False,** указывающих, какие элементы больше 5 |
| b= a[a > 5] | создание массива из наложения маски (>5) на массив a, в и ,elen все элементы a, отвечающие условию |
|  |  |
| v = [1,0,1]  vv = np.**tile**(v, (4,1)) | broadcasting:  сделать **массив большой из повторения маленького** (уложить плиточкой в виде маленького массива), указав количество строк и повторений в каждой строке |
| a[:, np.newaxis] | создать новое измерение и по всем измерениям разделить содержимое массива |
|  |  |
| ax, bx, cx = np**.ix\_(**a,b,c)  ax + bx \* cx | ix\_ функция для действий между массивами разного размера (количества элементов)  С преобразованными массивами можно производить поэлементные действия |
|  |  |
|  |  |
|  |  |
| **Pandas** |  |
| s = pd.Series(range(7, 16)) | создаем массив (Series) с данными из диапазона 7 включ-но -16 не включительно |
| s2 = pd.Series(data, index=[ascii\_lowercase[i] for i in data]) | создаем словарь данных из массива data с индексами в виде букв |
| s3.index = [ascii\_uppercase[i] for i in range(10)] | назначить индексами другие символы |
| s2['a'] | выбор значения по ключу |
| s2[['a', 'd', 'g']] = 13 | присваиваем значение 13 ячейкам под указанными ключами |
| s4.median()  s4.max()  s4.min() | показать среднее, максимальное и минимальное значения Series |
| s4.name = 'Numbers'  s4.index.name = 'letters' | Присвоить имена столбцам значений и индексов |
|  |  |
| df = pd.DataFrame({  'country': ['Russia', 'Kazakhstan', 'Ukraine', 'Belarus'],  'population': [144.5, 18.3, 42.4, 9.5],  'square': [17\_125\_200, 2\_724\_900, 603\_628, 207\_595],  }, index=['RU', 'KZ', 'UA', 'BY']) | Создание таблицы-ДатаФрейма с именами столбцов и индексами в виде кодов стран |
| df.head()  df.tail() | Показать первые или последние 5 элементов ДФ |
| df['population'] | выбор столбца Датафрейма по его имени |
| df.loc['KZ'] | выбор строки Датафрейма по её имени |
| df.loc['KZ']['population'] | выбор ячейки датафрейма по именам строки и столбца |
|  |  |
| df.loc['KZ':, 'square'] | **Срез** Датафрейма от строки Казахстана до конца и по столбцу square |
| df.iloc[:3] | **Срез** Датафрейма по ИНДЕКСУ строки от начала до 3 элемента не включительно |
|  |  |
| df[df.population > 42][['country', 'square']] | **Отбор из Датафрейма строк**, где колонка ‘population’ содержит значение больше 42. Для них показать срез колонок ‘country’ и ‘square’ |
|  |  |
| df['density'] = df.population / df.square \* 1\_000\_000 | **Добавление колонки** с просчётом в ней математического выражения на основании других колонок |
| df.drop(['density'], axis='columns') | Показать Датафрейм **без колонки** ‘density’ |
| del df['<ColumnName>'] | **Удалить** колонку |
| df.append(pd.Series(df.sum(), name='Full summary')) | **Добавить строку** со значением, равным сумме значений верхних строк и именем ‘Full summary’ |
|  |  |
|  |  |
| df.groupby(['<Column1Name>','<Column2Name>'])['<Uniqe Column>'].**count**() | подсчитать **количество** по уникальной колонке, сгруппировав по двум признакам (колонки 1 и 2) |
| pvt = titanic.pivot\_table(  index=['Sex'],  columns=['PClass'],  values='Name',  aggfunc='count'  ) | **Создать группировку** по полу, классу и подсчитать количество в данных группах |
| df.agg(‘sum’) | **показать суммы** значений по всем колонкам |
| df.agg([‘min’,’max’]) | показать минимумы и максимумы по всем колонкам |
| df.idxmax() | показать индексы максимальных значений для каждой колонки |
| df.loc[df.idxmax()] | показать максимальные значения в каждой колонке |
|  |  |
| summary = df.groupby('Name')['Fine sum'].**agg**(  ['count', 'sum']).**sort\_values**(['count'], ascending=False) | **Группировка** штрафов из ДФ по названию, вывод на экран суммы штрафов, обработанной функциями подсчета количества и суммы всех значений. Сортировка - по количеству. По убыванию (не по возрастанию) |
|  |  |
| df.to\_csv('countries\_data.csv') | Выгрузить данные в csv файл с указанием имени файла |
| pd.read\_csv('countries\_data.csv')  pd.read\_csv('countries\_data.csv', index\_col=0) | Прочитать данные из csv-файла с колонкой индексов или без неё |
| user1 = pd.read\_csv('dataset/1.csv', **names**=['TIME', 'X', 'Y', 'Z'], header=None) | Создать Data Frame из csv-файла **с указанием имён колонок**. |
| with open('fines.json') as f:  json\_str = f.read()  data\_json = json.loads(json\_str)['Value']  fines\_list = json.loads(data\_json)['Fines']  for fine in fines\_list:  fines.append({  'Name': fine['ApnDetail'][0]['Value'].replace('\t', ' - '),  'Place': fine['ApnDetail'][3]['Value'],  'Fine sum': fine['FineSum'],  }) | Расшифровка многоуровнего Json-файла до простого словаря: считываем из файла, выгружаем значения. Из значений - штрафы.  Потом заменяем сивол каретки на тире. |
| fines\_df = pd.DataFrame.from\_dict(fines) | Создать Датафрейм из словаря |
|  |  |
| df[‘City’].value\_counts(dropna=False) | Посчитать, сколько в данном столбце каждого вида значений. |
| df[‘City’].fillna(value=’VARIOUS’, inplace=True] | Вставить в пустые места столбца конкретные значения. |
| df[‘City’].fillna(df[‘City’].median(), inplace=True)  df[‘City’].fillna(df[‘City’].mean(), inplace=True) | Вставить в пустые места столбца средневзвешенное значение (медиану) этого же столбца. Или просто среднее арифметическое. |
|  |  |
| df['Sex'].**replace**(['male', 'female'], [1,0], inplace=True) | **Заменить данные в столбце** DataFrame  inplace = True заменит значения в указанном df, False - остваит df нетронутым, но создаст новый DataFrame |
| df.iat[0,4] = 1 | **Установить конкретное значение** в ячейке указанной по индексам |
| df.at[‘Diana’, ’Sex’] = 0 | **Установить конкретное значение** в ячейке указанной по **именам** строки и колонки |
|  |  |
| df.isnull().sum() | Проверяет, сколько в каждом столбце пропущенных значений (NaN) |
| df.dropna(how=’any’)  df.dropna(how=’all’) | Выкинуть строки, где хотя бы одно значение NaN / где ВСЕ значения пустые. |
| df.dropna(subset=[’City’, ‘Color’], how=’any’) | Выкинуть строки, где пустые значения есть в указанных столбцах. |
|  |  |
| df.City.astype(int) | **Сменить формат** столбца на заданный |
| df.price.str.replace(‘$’, ‘’).astype(float).mean() | **Меняем формат** «валюта» (currency, decimal) на float в столбце “price”, переходя на функции строк, выдёргивая знак доллара, затем меняя формат. Заодно считаем среднюю цену. |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
| **SKLearn** |  |
| s = (data.dtypes == 'object')  object\_cols = list(s[s].index) | Найти колонки с данными типа “object” (обычно str, categorical variables) |
| object\_cols = [col for col in X\_train.columns if X\_train[col].dtype == "object"] |
| good\_label\_cols = [col for col in object\_cols if  set(X\_train[col]) == set(X\_valid[col])] | Проверка совпадения значений колонок тренировочной и вввалидационной базы данных |
| bad\_label\_cols = list(set(object\_cols)-set(good\_label\_cols)) | # Problematic columns that will be dropped from the dataset |
| clean\_X\_train = X\_train.select\_dtypes(exclude=['object']) | 1. Удалить колонки с данными типа “object” |
| from sklearn.preprocessing import LabelEncoder  label\_encoder = LabelEncoder()  for col in object\_cols:  label\_X\_train[col] = label\_encoder.fit\_transform(X\_train[col])  label\_X\_valid[col] = label\_encoder.transform(X\_valid[col]) | 1. Применить встроенную sklearn функцию, меняющую данные object на int-заменители. |
| from sklearn.preprocessing import OneHotEncoder  OH\_encoder = OneHotEncoder(handle\_unknown='ignore', sparse=False)  OH\_cols\_train = pd.DataFrame(OH\_encoder.fit\_transform(X\_train[object\_cols]))  OH\_cols\_valid = pd.DataFrame(OH\_encoder.transform(X\_valid[object\_cols]))  OH\_cols\_train.index = X\_train.index  OH\_cols\_valid.index = X\_valid.index  num\_X\_train = X\_train.drop(object\_cols, axis=1)  num\_X\_valid = X\_valid.drop(object\_cols, axis=1)  OH\_X\_train = pd.concat([num\_X\_train, OH\_cols\_train], axis=1)  OH\_X\_valid = pd.concat([num\_X\_valid, OH\_cols\_valid], axis=1) | 1. применить функцию, **вставляющую вмето object несолько столбцов, по 1 на каждое значение** столбца с categorical variables   # Apply one-hot encoder to each column with categorical data  # One-hot encoding removed index; put it back  # Remove categorical columns (will replace with one-hot encoding)  # Add one-hot encoded columns to numerical features |
|  |  |
|  |  |
|  |  |