PANDAS: DATA MANIPULATION

**It is often said that 80% of data analysis is spent on the cleaning and small, but important,**

**aspect of data manipulation and cleaning with Pandas. ☺**

**Data structures**

• **Series** is a one-dimensional labeled array capable of holding any data type (integers, strings, floating point numbers, Python objects, etc.). The axis labels are collectively referred to as the index. The basic method to create a Series is to call ***pd.Series***

• **DataFrame** is a 2-dimensional labeled data structure with columns of potentially different types. You can think of it like a spreadsheet or SQL table, or a dict of Series objects. It stems from the *R* ***data.frame()***object.

|  |  |
| --- | --- |
| 1  2  3  4 | **from** **\_\_future\_\_** **import** print\_function  **import** **pandas** **as** **pd**  **import** **numpy** **as** **np**  **import** **matplotlib.pyplot** **as** **plt** |



**Create Data Frame :**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | #create a data frame  columns = ['name', 'age', 'gender', 'job']  user1 = pd.DataFrame([['alice', 19, "F", "student"],  ['john', 26, "M", "student"]],  columns=columns)  user2 = pd.DataFrame([['eric', 22, "M", "student"],  ['paul', 58, "F", "manager"]],  columns=columns)  user3 = pd.DataFrame(dict(name=['peter', 'julie'],  age=[33, 44], gender=['M', 'F'],  job=['engineer', 'scientist']))  **print**(user3) |

**Out Put:**

|  |
| --- |
| name age gender job |
| 0 peter 33 M engineer |
| 1 julie 44 F scientist |

**Combining Data Frames :**

|  |  |
| --- | --- |
| 1  2  3  4 | # Combining data frames  user1.append(user2)  users = pd.concat([user1, user2, user3])  **print**(users) |

**Out Put :**

|  |
| --- |
| name age gender job  0 alice 19 F student  1 john 26 M student  0 eric 22 M student  1 paul 58 F manager  0 peter 33 M engineer  1 julie 44 F scientist |

**Join Data Frame :**

|  |  |
| --- | --- |
| 1  2  3  4 | ## Join data frames  user4 = pd.DataFrame(dict(name=['alice', 'john', 'eric', 'julie'],  height=[165, 180, 175, 171]))  **print**(user4) |

**Out Put :**

|  |
| --- |
| name height  0 alice 165  1 john 180  2 eric 175  3 julie 171 |

Use intersection of keys from both frames

|  |  |
| --- | --- |
| 1  2  3  4 | ##Use intersection of keys from both frames  merge\_inter = pd.merge(users, user4, on="name")  **print**(merge\_inter) |

**Out Put :**

|  |
| --- |
| name age gender job height |
| 0 alice 19 F student 165 |
| 1 john 26 M student 180 |
| 2 eric 22 M student 175 |
| 3 julie 44 F scientist 171 |

**Reshaping by pivoting :**

**“Unpivots” a DataFrame from wide format to long (stacked) format,**

|  |  |
| --- | --- |
| 1  2  3  4  5 | #Reshaping by pivoting  #“Unpivots” a DataFrame from wide format to long (stacked) format,  staked = pd.melt(users, id\_vars="name", var\_name="variable", value\_name="value")  **print**(staked) |

**Out Put :**

|  |
| --- |
| name variable value |
| 0 alice age 19 |
| 1 john age 26 |
| 2 eric age 22 |
| 3 paul age 58 |
| 4 peter age 33 |
| 5 julie age 44 |
| 6 alice gender F |
| 7 john gender M |
| 8 eric gender M |
| 9 paul gender F |
| 10 peter gender M |
| 11 julie gender F |
| 12 alice job student |
| 13 john job student |
| 14 eric job student |
| 15 paul job manager |
| 16 peter job engineer |
| 17 julie job scientist |

**“pivots” a DataFrame from long (stacked) format to wide format,**

|  |  |
| --- | --- |
| 1 | **print**(staked.pivot(index='name', columns='variable', values='value')) |

**Out Put :**

|  |
| --- |
| variable age gender job  name  alice **19** F student  eric **22** M student  john **26** M student  julie **44** F scientist  paul **58** F manager  peter **33** M engineer |

**Summarizing :**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | *#Summarizing*  *# examine the users data*  users *# print the first 30 and last 30 rows*  type(users) *# DataFrame*  users.head() *# print the first 5 rows*  users.tail() *# print the last 5 rows*  users.index *# "the index" (aka "the labels")*  users.columns *# column names (which is "an index")*  users.dtypes *# data types of each column*  users.shape *# number of rows and columns*  users.values *# underlying numpy array*  users.info() *# concise summary (includes memory usage as of pandas 0.15.0)* |

**Out Put :**

|  |
| --- |
| <**class** 'pandas.core.frame.DataFrame'>  Int64Index: 6 entries, 0 to 1  Data columns (total 4 columns):  *# Column Non-Null Count Dtype*  --- ------ -------------- -----  0 name 6 non-null object  1 age 6 non-null int64  2 gender 6 non-null object  3 job 6 non-null object  dtypes: int64(1), object(3)  memory usage: 240.0+ bytes |

**Columns selection :**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | *##Columns selection*  users['gender'] *# select one column*  type(users['gender']) *# Series*  users.gender *# select one column using the DataFrame*  *# select multiple columns*  users[['age', 'gender']] *# select two columns*  my\_cols = ['age', 'gender'] *# or, create a list...*  users[my\_cols] *# ...and use that list to select columns*  type(users[my\_cols]) *# DataFrame* |

**Out Put :**

|  |  |
| --- | --- |
|  | **pandas.core.frame.DataFrame** |

**Rows selection (basic) :**

iloc is strictly integer position based,

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | *##Rows selection (basic)*  *## iloc is strictly integer position based*  df = users.copy()  df.iloc[0] *# first row*  df.iloc[0, 0] *# first item of first row*  df.iloc[0, 0] = 55  **for** i **in** range(users.shape[0]):  row = df.iloc[i]  row.age \*= 100 *# setting a copy, and not the original frame data.*  **print**(df) *# df is not modified* |

**Out Put :**

|  |  |
| --- | --- |
|  | name age gender job height  0 55 19 F student 165.0  1 john 26 M student 180.0  2 eric 22 M student 175.0  3 paul 58 F manager NaN  4 peter 33 M engineer NaN  5 julie 44 F scientist 171.0 |

**Sorting :**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | *## Sorting*  df = users.copy()  df.age.sort\_values() *# only works for a Series*  df.sort\_values(by='age') *# sort rows by a specific column*  df.sort\_values(by='age', ascending=False) *# use descending order instead*  df.sort\_values(by=['job', 'age']) *# sort by multiple columns*  df.sort\_values(by=['job', 'age'], inplace=True) *# modify df*  **print**(df) |

**Out Put :**

|  |
| --- |
| name age gender job  0 peter 33 M engineer  1 paul 58 F manager  1 julie 44 F scientist  0 alice 19 F student  0 eric 22 M student  1 john 26 M student |

**Descriptive statistics :**

|  |  |
| --- | --- |
| 1  2  3 | *## Descriptive statistics*  **print**(df.describe()) |

**Out Put :**

|  |
| --- |
| age  count 6.000000  mean 33.666667  std 14.895189  min 19.000000  25% 23.000000  50% 29.500000  75% 41.250000  max 58.000000 |

