DVA_lab-10_205229118_Mahalakshmi.S

May 22, 2021

0.0.1 Lab10. Advanced Data Wrangling in Panda

Import necessary modules

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

The data of the table:

[2]:
```
excelample = pd.DataFrame({'Month': ["January", "January", "January",␣
↪"January", "February", "February", "February", "February", "March", "March",␣
↪"March", "March"], 'Category': ["Transportation", "Grocery", "Household",␣
↪"Entertainment", "Transportation", "Grocery", "Household", "Entertainment",
"Transportation", "Grocery", "Household", "Entertainment"], 'Amount': [74.,␣
↪235., 175., 100., 115., 240., 225., 125., 90., 260., 200., 120.]})
```

[3]:
```
excelample
```

[3]:

	Month	Category	Amount
0	January	Transportation	74.0
1	January	Grocery	235.0
2	January	Household	175.0
3	January	Entertainment	100.0
4	February	Transportation	115.0
5	February	Grocery	240.0
6	February	Household	225.0
7	February	Entertainment	125.0
8	March	Transportation	90.0
9	March	Grocery	260.0
10	March	Household	200.0
11	March	Entertainment	120.0

[4]:
```
excelample_pivot = excelample.pivot(index="Category", columns="Month", values=
"Amount")
excelample_pivot
```

[4]:

Month	February	January	March
Category			
Entertainment	125.0	100.0	120.0
Grocery	240.0	235.0	260.0

```
Household          225.0    175.0  200.0
Transportation     115.0     74.0   90.0
```

Interested in Grand totals?

[5]:
```
# sum columns
excelample_pivot.sum(axis=1)
```

[5]:
```
Category
Entertainment     345.0
Grocery           735.0
Household         600.0
Transportation    279.0
dtype: float64
```

[6]:
```
# sum rows
excelample_pivot.sum(axis=0)
```

[6]:
```
Month
February    705.0
January     584.0
March       670.0
dtype: float64
```

0.0.2 Pivot is just reordering your data

Small subsample of the titanic dataset:

[7]:
```
df = pd.DataFrame({'Fare': [7.25, 71.2833, 51.8625, 30.0708, 7.8542, 13.0],
 'Pclass': [3, 1, 1, 2, 3, 2],
 'Sex': ['male', 'female', 'male', 'female', 'female', 'male'],
 'Survived': [0, 1, 0, 1, 0, 1]})
```

[8]:
```
df
```

[8]:

	Fare	Pclass	Sex	Survived
0	7.2500	3	male	0
1	71.2833	1	female	1
2	51.8625	1	male	0
3	30.0708	2	female	1
4	7.8542	3	female	0
5	13.0000	2	male	1

[9]:
```
df.pivot(index='Pclass', columns='Sex', values='Fare')
```

[9]:

Sex	female	male
Pclass		
1	71.2833	51.8625
2	30.0708	13.0000

```
3        7.8542   7.2500
```

0.0.3 Exercise: Create a Pivot table with 'Survided' values for Pclass vs Sex.

[11]:
```
table = pd.
↪pivot_table(df,index=['Pclass'],columns=['Sex'],values=['Survived'],aggfunc=np.
↪sum)
table
```

[11]:
```
        Survived
Sex       female male
Pclass
1              1    0
2              1    1
3              0    0
```

0.0.4 Let's now use the full Titanic Dataset

[12]:
```
import matplotlib.pyplot as plt
import seaborn as sns
df = sns.load_dataset('titanic') # avaiable inbuilt with seaborn
```

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

[13]:
```
   survived  pclass     sex   age  sibsp  parch     fare embarked  class  \
0         0       3    male  22.0      1      0   7.2500        S  Third
1         1       1  female  38.0      1      0  71.2833        C  First
2         1       3  female  26.0      0      0   7.9250        S  Third
3         1       1  female  35.0      1      0  53.1000        S  First
4         0       3    male  35.0      0      0   8.0500        S  Third

     who  adult_male deck  embark_town alive  alone
0    man        True  NaN  Southampton    no  False
1  woman       False    C    Cherbourg   yes  False
2  woman       False  NaN  Southampton   yes   True
3  woman       False    C  Southampton   yes  False
4    man        True  NaN  Southampton    no   True
```

And try the same pivot (no worries about the try-except, this is here just used to catch a loooong error):

[14]:
```
try:
    df.pivot(index='sex', columns='pclass', values='fare')
except Exception as e:
    print("Exception!", e)
```

```
Exception! Index contains duplicate entries, cannot reshape
```

This does not work, because we would end up with multiple values for one cell of the resulting frame, as the error says: duplicated values for the columns in the selection. As an example, consider the following rows of our three columns of interest:

[15]:
```
df.loc[[1, 3], ["sex", 'pclass', 'fare']]
```

[15]:
```
      sex  pclass     fare
1  female       1  71.2833
3  female       1  53.1000
```

Since pivot is just restructering data, where would both values of Fare for the same combination of Sex and Pclass need to go?

Well, they need to be combined, according to an aggregation functionality, which is supported by the function pivot_table

0.0.5 Pivot Tables - Aggregating while Pivoting

Pivot Table is a multidimensional version of GroupBy aggregation.

[16]:
```
df.pivot_table(index='sex', columns='pclass', values='fare')
```

[16]:
```
pclass           1          2          3
sex
female  106.125798  21.970121  16.118810
male     67.226127  19.741782  12.661633
```

Create a Pivot table with maximum 'fare' values for 'sex' vs 'pclass' columns

[17]:
```
df.pivot_table(index='sex', columns='pclass', values='fare', aggfunc='max')
```

[17]:
```
pclass         1     2      3
sex
female  512.3292  65.0  69.55
male    512.3292  73.5  69.55
```

Exercise: Create a Pivot table with the count of 'fare' values for 'sex' vs 'pclass' columns

[18]:
```
df.pivot_table(index='sex', columns='pclass', values='fare', aggfunc='count')
```

[18]:
```
pclass    1    2    3
sex
female   94   76  144
male    122  108  347
```

[19]:
```
pd.crosstab(index=df['sex'], columns=df['pclass'])
```

[19]:
```
pclass    1    2    3
sex
female   94   76  144
male    122  108  347
```

Exercise: Make a pivot table with the mean survival rates for pclass vs sex

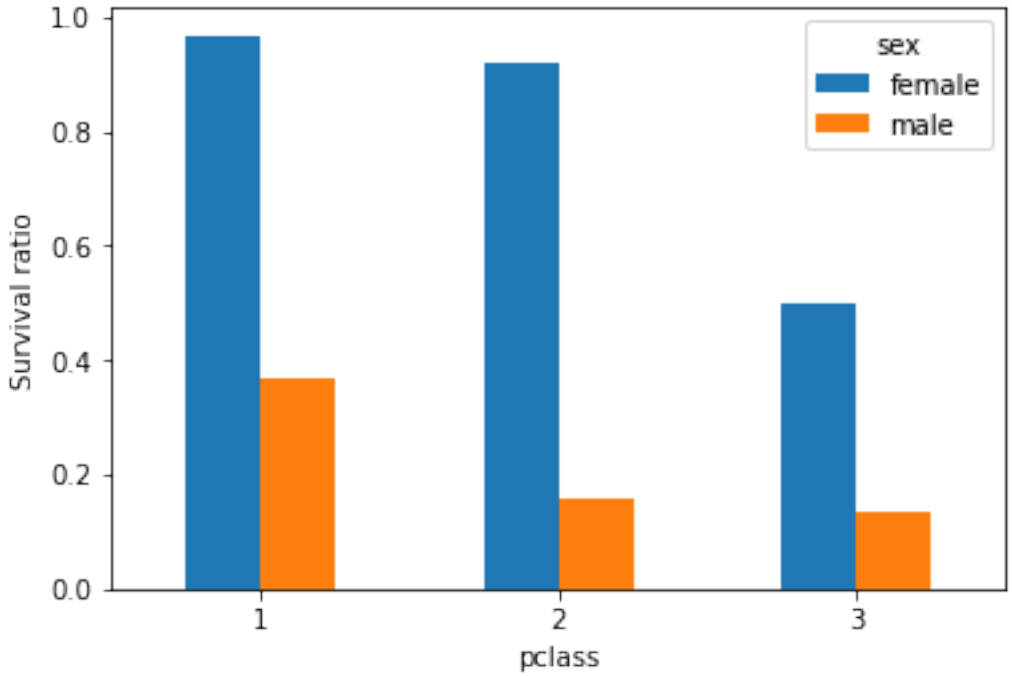
[21]:
```
df.pivot_table(index='sex', columns='pclass', values='survived', aggfunc='mean')
```

[21]:
```
pclass         1         2         3
sex
female  0.968085  0.921053  0.500000
male    0.368852  0.157407  0.135447
```

Plot Bar Chart for Survival ratio

[22]:
```
fig, ax1 = plt.subplots()
df.pivot_table(index='pclass', columns='sex', values='survived',␣
 ↪aggfunc='mean').plot(kind='bar', rot=0, ax=ax1)
ax1.set_ylabel('Survival ratio')
```

[22]:
```
Text(0, 0.5, 'Survival ratio')
```



Exercise: Make a pivot table of the median Fare payed by aged vs sex

[25]:
```
median_age_table = df.pivot_table(index='age', columns='sex',values='fare',␣
 ↪aggfunc='median')
```

[28]:
```
# let us show only 5 rows
median_age_table.head()
```

[28]:
```
sex    female      male
age
0.42      NaN    8.5167
0.67      NaN   14.5000
0.75  19.2583       NaN
0.83      NaN   23.8750
0.92      NaN  151.5500
```

Exercise: Make a pivot table of the median Fare payed by 'underaged' vs 'sex'

[29]:
```
# Create a new column 'underaged' and store the result of the condition age <=
 ↪18
df['underaged'] = df["age"]<=18
```

[35]:
```
df
```

[35]:
```
     survived  pclass     sex   age  sibsp  parch     fare embarked   class  \
0           0       3    male  22.0      1      0   7.2500        S   Third
1           1       1  female  38.0      1      0  71.2833        C   First
2           1       3  female  26.0      0      0   7.9250        S   Third
3           1       1  female  35.0      1      0  53.1000        S   First
4           0       3    male  35.0      0      0   8.0500        S   Third
..        …     …      …   …    …     …       …      …      …
886         0       2    male  27.0      0      0  13.0000        S  Second
887         1       1  female  19.0      0      0  30.0000        S   First
888         0       3  female   NaN      1      2  23.4500        S   Third
889         1       1    male  26.0      0      0  30.0000        C   First
890         0       3    male  32.0      0      0   7.7500        Q   Third

       who  adult_male deck  embark_town alive  alone  underaged
0      man        True  NaN  Southampton    no  False      False
1    woman       False    C    Cherbourg   yes  False      False
2    woman       False  NaN  Southampton   yes   True      False
3    woman       False    C  Southampton   yes  False      False
4      man        True  NaN  Southampton    no   True      False
..     …          …  …          …    …     …        …
886    man        True  NaN  Southampton    no   True      False
887  woman       False    B  Southampton   yes   True      False
888  woman       False  NaN  Southampton    no  False      False
889    man        True    C    Cherbourg   yes   True      False
890    man        True  NaN   Queenstown    no   True      False

[891 rows x 16 columns]
```

[36]:
```
# Now, make the pivot table for underaged
median_age_table=df.pivot_table(index='underaged', columns='sex',values='fare',
 ↪aggfunc='median')
```

[37]:
```
median_age_table
```

[37]:
```
sex         female     male
underaged
False      24.1500  10.3354
True       20.2875  20.2500
```

0.0.6 Grouping Pivot table

[31]:
```
age = pd.cut(df['age'], [0, 18, 80])
df.pivot_table('survived', ['sex', age], 'class')
```

[31]:
```
class               First    Second     Third
sex    age
female (0, 18]   0.909091  1.000000  0.511628
       (18, 80]  0.972973  0.900000  0.423729
male   (0, 18]   0.800000  0.600000  0.215686
       (18, 80]  0.375000  0.071429  0.133663
```

We can apply this same strategy when working with the columns as well; let's add info on the fare paid using pd.qcut to automatically compute quantiles

[32]:
```
fare = pd.qcut(df['fare'], 2)
df.pivot_table('survived', ['sex', age], [fare, 'class'])
```

[32]:
```
fare            (-0.001, 14.454]                    (14.454, 512.329]  \
class                      First    Second     Third             First
sex    age
female (0, 18]               NaN  1.000000  0.714286          0.909091
       (18, 80]              NaN  0.880000  0.444444          0.972973
male   (0, 18]               NaN  0.000000  0.260870          0.800000
       (18, 80]              0.0  0.098039  0.125000          0.391304

fare
class                Second     Third
sex    age
female (0, 18]   1.000000  0.318182
       (18, 80]  0.914286  0.391304
male   (0, 18]   0.818182  0.178571
       (18, 80]  0.030303  0.192308
```

The result is a four-dimensional aggregation with hierarchical indices

0.0.7 Multiple Aggregate Functions

[33]:
```
df.pivot_table(index='sex', columns='class',
aggfunc={'survived':sum, 'fare':'mean'})
```

[33]:
```
               fare                          survived
class         First     Second      Third    First Second Third
sex
female   106.125798  21.970121  16.118810       91     70    72
male      67.226127  19.741782  12.661633       45     17    47
```

0.0.8 Melt - from Pivot Table to long or tidy format

The melt function performs the inverse operation of a pivot . This can be used to make your frame longer, i.e. to make a tidy version of your data.

[38]:
```
pivoted = df.pivot_table(index='sex', columns='pclass', values='fare').
↪reset_index()
pivoted.columns.name = None
```

[39]:
```
pivoted
```

[39]:
```
      sex           1          2          3
0  female  106.125798  21.970121  16.118810
1    male   67.226127  19.741782  12.661633
```

Assume we have a DataFrame like the above. The observations (the average Fare people payed) are spread over different columns. In a tidy dataset, each observation is stored in one row. To obtain this, we can use the melt function:

[40]:
```
pd.melt(pivoted)
```

[40]:
```
   variable    value
0       sex   female
1       sex     male
2         1  106.126
3         1  67.2261
4         2  21.9701
5         2  19.7418
6         3  16.1188
7         3  12.6616
```

As you can see above, the melt function puts all column labels in one column, and all values in a second column.

In this case, this is not fully what we want. We would like to keep the 'Sex' column separately:

[41]:
```
pd.melt(pivoted, id_vars=['sex']) #, var_name='pclass', value_name='fare')
```

[41]:
```
      sex variable       value
0  female        1  106.125798
1    male        1   67.226127
2  female        2   21.970121
3    male        2   19.741782
```

```
4  female        3   16.118810
5    male        3   12.661633
```

0.0.9 Reshaping with stack and unstack

Before we speak about hierarchical index , first check it in practice on the following dummy example:

[42]:
```
df2 = pd.DataFrame({'A':['one', 'one', 'two', 'two'],
 'B':['a', 'b', 'a', 'b'],
 'C':range(4)})
df2
```

[42]:
```
     A  B  C
0  one  a  0
1  one  b  1
2  two  a  2
3  two  b  3
```

To use stack / unstack , we need the values we want to shift from rows to columns or the other way around as the index:

[43]:
```
df2 = df2.set_index(['A', 'B']) # Indeed, you can combine two indices
df2
```

[43]:
```
       C
A   B
one a  0
    b  1
two a  2
    b  3
```

[44]:
```
result = df2['C'].unstack()
result
```

[44]:
```
B    a  b
A
one  0  1
two  2  3
```

[45]:
```
df2 = result.stack().reset_index(name='C')
df2
```

[45]:
```
     A  B  C
0  one  a  0
1  one  b  1
2  two  a  2
3  two  b  3
```

0.0.10 Mimick Pivot Table

To better understand and reason about pivot tables, we can express this method as a combination of more basic steps. In short, the pivot is a convenient way of expressing the combination of a groupby and stack/unstack .

Let us come back to our titanic dataset

[46]:
```
df.head()
```

[46]:
```
   survived  pclass     sex   age  sibsp  parch     fare embarked  class  \
0         0       3    male  22.0      1      0   7.2500        S  Third
1         1       1  female  38.0      1      0  71.2833        C  First
2         1       3  female  26.0      0      0   7.9250        S  Third
3         1       1  female  35.0      1      0  53.1000        S  First
4         0       3    male  35.0      0      0   8.0500        S  Third

     who  adult_male deck  embark_town alive  alone  underaged
0    man        True  NaN  Southampton    no  False      False
1  woman       False    C    Cherbourg   yes  False      False
2  woman       False  NaN  Southampton   yes   True      False
3  woman       False    C  Southampton   yes  False      False
4    man        True  NaN  Southampton    no   True      False
```

[47]:
```
df.pivot_table(index='pclass', columns='sex', values='survived', aggfunc='mean')
```

[47]:
```
sex       female      male
pclass
1       0.968085  0.368852
2       0.921053  0.157407
3       0.500000  0.135447
```

0.0.11 Exercise:

- Get the same result as above based on a combination of `groupby` and `unstack`
- First use `groupby` to calculate the survival ratio for all groups`unstack`
- Then, use `unstack` to reshape the output of the groupby operation

[49]:
```
unst=df.groupby(['pclass','sex'])['survived'].agg('mean')
unst.unstack()
```

[49]:
```
sex       female      male
pclass
1       0.968085  0.368852
2       0.921053  0.157407
3       0.500000  0.135447
```