Department of Data Science - Data and Visual Analytics Lab

Lab10. Advanced Data Wrangling in Pandas

Objectives ¶

After completing this lab, you will be able to create and apply some advanced features of Pandas including

```
- pivot table
- crosstab
- cut and qcut
- melt
- stack and unstack
```

Import necessary modules

In [1]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt.
```

Pivoting data in MS Excel

People who know Excel, probably know the **Pivot** functionality:

	A	B	C
1	MONTH	CATEGORY	AMOUNT
2	January	Transportation	$74.00
3	January	Grocery	$235.00
4	January	Household	$175.00
5	January	Entertainment	$100.00
6	February	Transportation	$115.00
7	February	Grocery	$240.00
8	February	Household	$225.00
9	February	Entertainment	$125.00
10	March	Transportation	$90.00
11	March	Grocery	$260.00
12	March	Household	$200.00
13	March	Entertainment	$120.00

Sum of AMOUNT	Column La			
Row Labels	January	February	March	Grand Total
Entertainment	$100	$125	$120	$345
Grocery	$235	$240	$260	$735
Household	$175	$225	$200	$600
Transportation	$74	$115	$90	$279
Grand Total	$584	$705	$670	$1,959

The data of the table:

In [2]:
```
excelample = pd.DataFrame({'Month': ["January", "January", "January", "Januar
y",
                                     "February", "February", "February", "Februar
y",
                                     "March", "March", "March", "March"],
                           'Category': ["Transportation", "Grocery", "Household", "Ent
ertainment",
                                    "Transportation", "Grocery", "Household", "Ent
ertainment",
                                    "Transportation", "Grocery", "Household", "Ent
ertainment"],
                           'Amount': [74., 235., 175., 100., 115., 240., 225., 125., 9
0., 260., 200., 120.]})
```

In [3]:
```
excelample
```

Out[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

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

Out[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*?

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

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

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

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

Pivot is just reordering your data

Small subsample of the titanic dataset:

In [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', 'mal
e'],
                   'Survived': [0, 1, 0, 1, 0, 1]})
```

In [8]:
```
df
```

Out[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

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

Sex	female	male
Pclass		
1	71.2833	51.8625
2	30.0708	13.0000
3	7.8542	7.2500

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

In [10]:
```
df. pivot (index = "pclass", columns="Sex", values="Survived")
```
Out[10]:

Sex	female	male
Pclass		
1	1	0
2	1	1
3	0	0

Let's now use the full Titanic Dataset

In [11]:
```
df = sns.load_dataset('titanic')  # avaiable inbuilt with seaborn
```
In [12]:
```
df.head()
```
Out[12]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True

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

In [13]:
```
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:

In [14]:
```
df.loc[[1, 3], ["sex", 'pclass', 'fare']]
```

Out[14]:

	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`

NOTE:

- Pivot is purely restructering: a single value for each index/column combination is required.

Pivot Tables - Aggregating while Pivoting

Pivot Table is a multidimensional version of GroupBy aggregation.

In [15]:
```
df.pivot_table(index='sex', columns='pclass', values='fare')
```

Out[15]:

pclass	1	2	3
sex			
female	106.125798	21.970121	16.118810
male	67.226127	19.741782	12.661633

REMEMBER: * By default, `pivot_table` takes the **mean** of all values that would end up into one cell. However, you can also specify other aggregation functions using the `aggfunc` keyword.

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

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

Out[16]:

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

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

Out[17]:

pclass	1	2	3
sex			
female	94	76	144
male	122	108	347

REMEMBER:

- There is a shortcut function for a `pivot_table` with a `aggfunc='count'` as aggregation: `crosstab`

```
In [18]: pd.crosstab(index=df['sex'], columns=df['pclass'])
```

Out[18]:

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

In [19]: df.pivot_table(index='sex', columns='pclass', values='survived', aggfunc='mean')

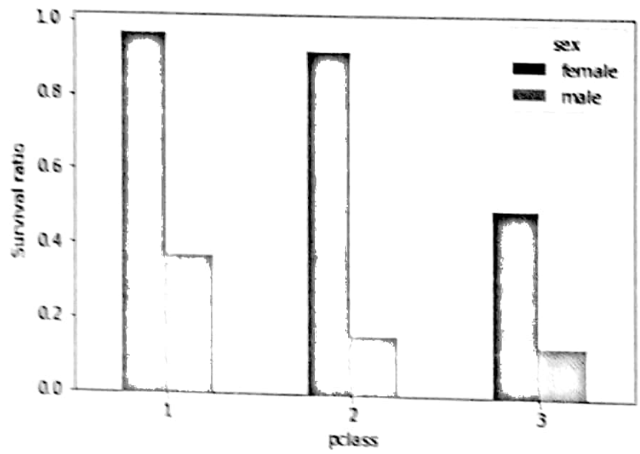
Out[19]:

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

In [20]:
```
fig, ax1 = plt.subplots()
df.pivot_table(index='pclass', columns='sex',
               values='survived', aggfunc='mean').plot(kind='bar', rot=0, ax=a
x1)
ax1.set_ylabel('Survival ratio')
```

Out[20]: Text(0, 0.5, 'Survival ratio')



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

Scanned with CamScanner

In [21]:
```
median_age_table = df.pivot_table(index='age', columns='sex', values='fare')

# Let us show only 5 rows
median_age_table[:5]
```

Out[21]:

sex	female	male
age		
0.42	NaN	8.5167
0.67	NaN	14.5000
0.75	19.25830	NaN
0.83	NaN	23.8750
0.92	NaN	151.5500
1.00	13.43750	39.0000
2.00	26.95000	27.5625
3.00	31.32710	22.3750
4.00	22.02500	29.1250
5.00	23.50415	NaN

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

In [22]:
```
# Create a new column 'underaged' and store the result of the condition age <= 18
df['underaged'] = df["age"] <=18
```

In [23]:
```
# Now, make the pivot table for underaged median_age_table
```

Out[23]:

sex	female	male
underaged		
False	24.1500	10.3354
True	20.2875	20.2500

Grouping Pivot table

```
In [24]: age = pd.cut(df['age'], [0, 18, 80])
         df.pivot_table('survived', ['sex', age], 'class')
```

Out[24]:

	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

```
In [25]: fare = pd.qcut(df['fare'], 2)
         df.pivot_table('survived', ['sex', age], [fare, 'class'])
```

Out[25]:

	fare	(-0.001, 14.454]			(14.454, 512.329]		
	class	**First**	**Second**	**Third**	**First**	**Second**	**Third**
sex	**age**						
female	**(0, 18]**	NaN	1.000000	0.714286	0.909091	1.000000	0.318182
	(18, 80]	NaN	0.880000	0.444444	0.972973	0.914286	0.391304
male	**(0, 18]**	NaN	0.000000	0.260870	0.800000	0.818182	0.178571
	(18, 80]	0.0	0.098039	0.125000	0.391304	0.030303	0.192308

The result is a four-dimensional aggregation with hierarchical indices

Multiple Aggregate Functions

```
In [26]: df.pivot_table(index='sex', columns='class',
         aggfunc={'survived':sum, 'fare':'mean'})
```

Out[26]:

	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

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.

In [27]:
```
pivoted = df.pivot_table(index='sex', columns='pclass', values='fare').reset_i
ndex()
pivoted.columns.name = None
```

In [28]:
```
pivoted
```

Out[28]:

	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:

In [29]:
```
pd.melt(pivoted)
```

Out[29]:

	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:

```
In [30]: pd.melt(pivoted, id_vars=['sex']) #, var_name='pclass', value_name='fare')
```

Out[30]:

	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

Reshaping with `stack` and `unstack`

The docs say:

> Pivot a level of the (possibly hierarchical) column labels, returning a DataFrame (or Series in the case of an object with a single level of column labels) having a hierarchical index with a new inner-most level of row labels.

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

```
In [31]: df2 = pd.DataFrame({'A':['one', 'one', 'two', 'two'],
                             'B':['a', 'b', 'a', 'b'],
                             'C':range(4)})
         df2
```

Out[31]:

	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:

In [32]:
```
df2 = df2.set_index(['A', 'B']) # Indeed, you can combine two indices
df2
```

Out[32]:

A	B	C
one	a	0
	b	1
two	a	2
	b	3

In [33]:
```
result = df2['C'].unstack()
result
```

Out[33]:

A \ B	a	b
one	0	1
two	2	3

In [34]:
```
df2 = result.stack().reset_index(name='C')
df2
```

Out[34]:

	A	B	C
0	one	a	0
1	one	b	1
2	two	a	2
3	two	b	3

REMEMBER:

- **stack**: make your data *longer* and *smaller*
- **unstack**: make your data *shorter* and *wider*

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

In [35]:
```
df.head()
```
Out[35]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True

In [36]:
```
df.pivot_table(index='pclass', columns='sex',
               values='survived', aggfunc='mean')
```
Out[36]:

sex	female	male
pclass		
1	0.968085	0.368852
2	0.921053	0.157407
3	0.500000	0.135447

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

In [37]:
```
temp = df.groupby(['pclass', 'sex'])['survived'].agg('mean')
temp.unstack()
```
Out[37]:

sex	female	male
pclass		
1	0.968085	0.368852
2	0.921053	0.157407
3	0.500000	0.135447

In [ ]: