Sane's STATS Academy of Statistics

More on pandas

Data Reshaping, Grouped Processing

Checking missing values

- For checking the missing values, we can use `pd.isnull()` / `pd.ina()`
- For checking non-missing values, we can use `pd.notnull()`

```
In [9]: job
Out[9]:
   S_No  Computer  Marketing  Engineer
0     1      84.0       55.0        75
1     2      90.0       53.0        70
2     3      63.0        NaN        88
3     4       NaN       50.0        77
4     5      63.0       41.0        89
5     6      75.0       54.0        85
6     7       NaN       51.0        96
7     8      77.0       64.0       105
```

```
In [10]: pd.isnull(job)
Out[10]:
    S_No  Computer  Marketing  Engineer
0  False     False      False     False
1  False     False      False     False
2  False     False       True     False
3  False      True      False     False
4  False     False      False     False
5  False     False      False     False
6  False      True      False     False
7  False     False      False     False
```

```
In [11]: pd.notnull(job)
Out[11]:
   S_No  Computer  Marketing  Engineer
0  True      True       True      True
1  True      True       True      True
2  True      True      False      True
3  True     False       True      True
4  True      True       True      True
5  True      True       True      True
6  True     False       True      True
7  True      True       True      True
```

Sane's STATS Academy of Statistics

Working with missing values

- Missing values can be imputed by some educated guess like mean imputation or median imputation

```
In [9]: job
Out[9]:
   S_No  Computer  Marketing  Engineer
0     1      84.0       55.0        75
1     2      90.0       53.0        70
2     3      63.0        NaN        88
3     4       NaN       50.0        77
4     5      63.0       41.0        89
5     6      75.0       54.0        85
6     7       NaN       51.0        96
7     8      77.0       64.0       105
```

```
In [12]: mu_comp = job['Computer'].mean()

In [13]: mu_comp = job['Computer'].mean()
    ...: job['a_comp'] = job['Computer'].fillna(mu_comp)
    ...: job
Out[13]:
   S_No  Computer  Marketing  Engineer     a_comp
0     1      84.0       55.0        75  84.000000
1     2      90.0       53.0        70  90.000000
2     3      63.0        NaN        88  63.000000
3     4       NaN       50.0        77  75.333333
4     5      63.0       41.0        89  63.000000
5     6      75.0       54.0        85  75.000000
6     7       NaN       51.0        96  75.333333
7     8      77.0       64.0       105  77.000000
```

Apply function on pandas

- We can apply the aggregation functions like `numpy.mean()` row-wise and column-wise

```
# Column-wise mean
boston.apply(np.mean, axis=0)

# Row-wise mean
boston.apply(np.mean, axis=1)
```

Melting the data

- The data is reshaped in by stacking its columns one below the other
- Function `pandas.melt()` can melt the data frame

```
In [25]: quality
Out[25]:
   Sno    A   B   C
0    1   97  93  99
1    2   73  14  94
2    3   93  93  87
3    4  100  55  66
4    5   23  77  59

In [26]: qual_melt = pd.melt(quality, id_vars='Sno')
```

```
In [27]: qual_melt
Out[27]:
    Sno variable  value
0     1        A     97
1     2        A     73
2     3        A     93
3     4        A    100
4     5        A     23
5     1        B     93
6     2        B     14
7     3        B     93
8     4        B     55
9     5        B     77
10    1        C     99
11    2        C     94
12    3        C     87
13    4        C     66
14    5        C     59
```

Sane's STATS Academy of Statistics

Pivot Table

- We can break the molten frame with the help of `pandas.pivot_table()`

```
In [28]: qual_pivot = pd.pivot_table(qual_melt, index='Sno',
    ...:                             columns='variable',values='value')

In [29]: qual_pivot
Out[29]:
variable    A   B   C
Sno
1          97  93  99
2          73  14  94
3          93  93  87
4         100  55  66
5          23  77  59
```

Group by

- We can call the function `groupby()` on the classification variable and then call a aggregate function on it

```
In [30]: qual_melt.groupby('variable')['value'].mean()
Out[30]:
variable
A    77.2
B    66.4
C    81.0
Name: value, dtype: float64

In [31]: qual_melt.groupby('variable')['value'].std()
Out[31]:
variable
A    32.081147
B    33.178306
C    17.592612
Name: value, dtype: float64
```

Frequency tables in **pandas**

Syntax : pandas.crosstab(index, columns, values=None, rownames=None, colnames=None, aggfunc=None, margins=False, margins_name='All', dropna=True, normalize=False)

Where

index : array-like, Series, or list of arrays/Series Values to group by in the rows
columns : array-like, Series, or list of arrays/Series
Values to group by in the columns
values : array-like, optional
Array of values to aggregate according to the factors. Requires aggfunc be specified.
aggfunc : function, optional
If specified, requires values be specified as well
rownames : sequence, default None
If passed, must match number of row arrays passed
colnames : sequence, default None
If passed, must match number of column arrays passed
margins : boolean, default False
Add row/column margins (subtotals)
margins_name : string, default 'All'

crosstab Examples

```
In [30]: pd.crosstab(index=telecom["Response"], columns="count")
Out[30]:
col_0     count
Response
N            72
Y            78

In [31]: pd.crosstab(index=telecom["Response"], columns=telecom["Gender"])
Out[31]:
Gender      F   M
Response
N          13  59
Y          64  14
```

Margin totals in **crosstab**

```
In [37]: pd.crosstab(index=telecom["Response"],columns=telecom["Gender"],margins=True)
Out[37]:
Gender      F   M  All
Response
N          13  59   72
Y          64  14   78
All        77  73  150
```