Pandas

Selection of data

- The indexing field ***ix*** enables to select a subset of the rows and columns from a DataFrame.

```
In: df
Out:
     Paris  Berlin  Madrid
b        0       1       2
a        3       7       5
c        6       4       8
```

```
In:df.ix['a','Berlin']
Out:
7
```

```
In: df.ix[['b','c'],'Berlin']
Out:
b     1
c     4
Name: Berlin
```

```
In:df.ix[:,'Berlin']
Out:
b     1
a     7
c     4
Name: Berlin
```

Aegis
SCHOOL OF BUSINESS
SCHOOL OF DATA SCIENCE
SCHOOL OF TELECOMMUNICATION

DataFrame ordering/sorting

- No order method for DataFrame: specify the axis

```
In: df
Out:
   Paris  Berlin  Madrid
b      0       1       2
a      3       7       5
c      6       4       8
```

```
In:df.sort_index([ascending=True])
Out:
   Paris  Berlin  Madrid
a      3       7       5
b      0       1       2
c      6       4       8
```

```
In:df.sort_index(by = 'Berlin')
Out:
       Paris  Berlin  Madrid
b      0       1       2
c      6       4       8
a      3       7       5
```

```
In: df.sort_index(axis=1)
Out:

   Berlin  Madrid  Paris
a      7       5      3
b      1       2      0
c      4       8      6
```

Computing Descriptive Statistics

- Objects are equipped with a set of common statistical methods.

```
In: df
Out:
    Paris  Berlin  Madrid
b       0       1       2
a       3       7       5
c       6       4       8
```

```
In: df.describe()
Out:
         Paris  Berlin  Madrid
count      3.0     3.0     3.0
mean       3.0     4.0     5.0
std        3.0     3.0     3.0
min        0.0     1.0     2.0
25%        1.5     2.5     3.5
50%        3.0     4.0     5.0
75%        4.5     5.5     6.5
max        6.0     7.0     8.0
```

```
In: df.sum(axis=1)
Out:
b     3
a    15
c    18
```

- Covariance and correlation

```
In: df.cov()
Out:
        Paris  Berlin  Madrid
Paris     9.0     4.5     9.0
Berlin    4.5     9.0     4.5
Madrid    9.0     4.5     9.0
```

```
In: df.corr()
Out:
        Paris  Berlin  Madrid
Paris     1.0     0.5     1.0
Berlin    0.5     1.0     0.5
Madrid    1.0     0.5     1.0
```

Function application

- Apply mathematical functions directly on values

```
In: df
Out :
    Paris   Berlin   Madrid
b       0        1        2
a       3        7        5
c       6        4        8
```

```
f = lambda x: math.sqrt(x)
In: df.applymap(f)
Out:
       Paris     Berlin     Madrid
b   0.000000   1.000000   1.414214
a   1.732051   2.645751   2.236068
c   2.449490   2.000000   2.828427
```

```
df.Berlin = df['Berlin'].map(f)

In: df
Out:
   Paris    Berlin    Madrid
b      0   1.000000      2
a      3   2.645751      5
c      6   2.000000      8
```

Exercise

Assign in a new column 'Total' the sum of the others columns amount values applied with the function f(x) = x+ 0.2*x and sort the table by total value

Concatenation

Concatenation in pandas is the process of either adding rows to the end of an existing Series or DataFrame object or adding additional columns to a DataFrame. In pandas, concatenation is performed via the pandas function pd.concat(). The function will perform the operation on a specific axis and as we will see, will also perform any required set logic involved in aligning along that axis.

```
# two Series objects to concatenate
s1 = pd.Series(np.arange(0, 3))
s2 = pd.Series(np.arange(5, 8))
```

```
# concatenate them
pd.concat([s1, s2])
```

```
0 0
1 1
2 2
0 5
1 6
2 7
dtype: int64
```

Aegis
SCHOOL OF BUSINESS
SCHOOL OF DATA SCIENCE
SCHOOL OF TELECOMMUNICATION

Data Munging in Python : Using Pandas

Data Munging

- While our exploration of the data, we found a few problems in the data set, which needs to be solved before the data is ready for a good model. This exercise is typically referred as "Data Munging".
- Here are the problems, we are already aware of:
- There are missing values in some variables. We should estimate those values wisely depending on the amount of missing values and the expected importance of variables.
- While looking at the distributions, we saw that ApplicantIncome and LoanAmount seemed to contain extreme values at either end. Though they might make intuitive sense, but should be treated appropriately.

NA handling methods

Pandas treats None and NaN as essentially interchangeable for indicating missing or null values. To facilitate this convention, there are several useful methods for detecting, removing, and replacing null values in Pandas data structures.

dropna(): return a filtered version of the data
Fillna():Fill in missing data with some value or using an interpolation method
isnull ():Return like-type object containing Boolean values indicating which values are missing / NA.
notnull(): opposite of isnull()

```
import pandas as pd
data = pd.Series([1, NA, 3.5, NA, 7])
```

```
0    1.0
1    NaN
2    3.5
3    NaN
4    7.0
dtype: float64
```

```
data.dropna()
```

```
0    1.0
2    3.5
4    7.0
dtype: float64
```

```
data.fillna(0)
```

```
0    1.0
1    0.0
2    3.5
3    0.0
4    7.0
```

```
data.isnull()
```

```
0    False
1     True
2    False
3     True
4    False
dtype: bool
```

```
data[data.notnull()]
```

```
0    1.0
2    3.5
4    7.0
dtype: float64
```

“However, when the sample size is large and does not include outliers, the **mean** score usually provides a better measure of central tendency.

The **median** is usually preferred to other measures of central tendency when your data set is skewed (i.e., forms a skewed distribution) or you are dealing with **ordinal** data

- df=pd.read_csv("D:/Python Dataset/train.csv") #Reading the dataset in a dataframe using Pandas
- In addition to these problems with numerical fields, we should also look at the non-numerical fields i.e. Gender, Property_Area, Married, Education and Dependents to see, if they contain any useful information.

Check missing values in the dataset

- Let us look at missing values in all the variables because most of the models don't work with missing data and even if they do, imputing them helps more often than not. So, let us check the number of nulls / NaNs in the dataset

```
df.apply(lambda x: sum(x.isnull()),axis=0)
```

This command should tell us the number of missing values in each column as isnull() returns 1, if the value is null.

```
In [14]: df.apply(lambda x: sum(x.isnull()),axis=0)

Out[14]: Loan_ID               0
         Gender               13
         Married               3
         Dependents           15
         Education             0
         Self_Employed        32
         ApplicantIncome       0
         CoapplicantIncome     0
         LoanAmount           22
         Loan_Amount_Term     14
         Credit_History       50
         Property_Area         0
         Loan_Status           0
         dtype: int64
```

- Though the missing values are not very high in number, but many variables have them and each one of these should be estimated and added in the data.
- Note: Remember that missing values may not always be NaNs.
- For instance, if the Loan_Amount_Term is 0, does it makes sense or would you consider that missing? I suppose your answer is missing and you’re right. So we should check for values which are unpractical.

How to fill missing values in LoanAmount?

- There are numerous ways to fill the missing values of loan amount – the simplest being replacement by mean, which can be done by following code:

  df['LoanAmount'].fillna(df['LoanAmount'].mean(), inplace=True)

The other extreme could be to build a supervised learning model to predict loan amount on the basis of other variables and then use age along with other variables to predict survival.

- Since, the purpose now is to bring out the steps in data munging, I'll rather take an approach, which lies some where in between these 2 extremes.
- A key hypothesis is that the whether a person is educated or self-employed can combine to give a good estimate of loan amount.

Aegis
SCHOOL OF BUSINESS
SCHOOL OF DATA SCIENCE
SCHOOL OF TELECOMMUNICATION

- As we say earlier, Self_Employed has some missing values. Let's look at the frequency table:

```
In [40]: df['Self_Employed'].value_counts()

Out[40]: No      500
         Yes      82
         Name: Self_Employed, dtype: int64
```

- Since ~86% values are "No", it is safe to impute the missing values as "No" as there is a high probability of success. This can be done using the following code:

```
df['Self_Employed'].fillna('No',inplace=True)
```

inplace : *boolean, default False*

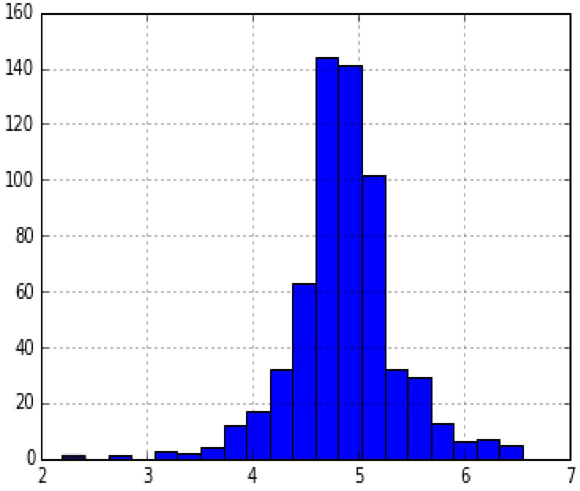
If True, in place. Note: this will modify any other views on this object (e.g. a column form a DataFrame). Returns the caller if this is True.

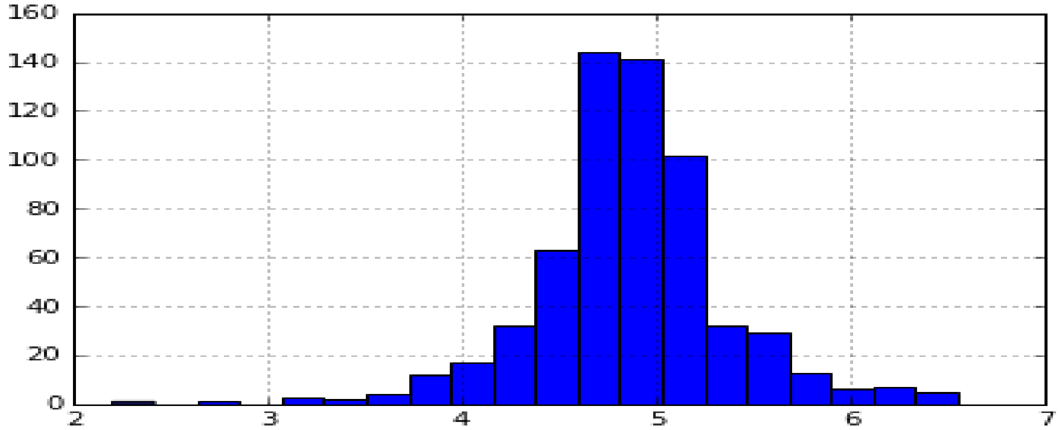
- `

Legis
SCHOOL OF BUSINESS
SCHOOL OF DATA SCIENCE
SCHOOL OF TELECOMMUNICATION

How to treat for extreme values in distribution of LoanAmount and ApplicantIncome?

- Let’s analyze LoanAmount first. Since the extreme values are practically possible, i.e. some people might apply for high value loans due to specific needs. So instead of treating them as outliers, let’s try a log transformation to nullify their effect:
- df['LoanAmount_log']=np.log(df['LoanAmount'])
  df['LoanAmount_log'].hist(bins=20)





- Now we see that the distribution is much better than before. I will leave it upto you to impute the missing values for Gender, Married, Dependents, Loan_Amount_Term, Credit_History.
- Also, I encourage you to think about possible additional information which can be derived from the data.
- For example, creating a column for LoanAmount/TotalIncome might make sense as it gives an idea of how well the applicant is suited to pay back his loan.
- Next, we will look at making predictive models.

```
import pandas as pd
import numpy as np
import matplotlib as plt
df = pd.read_csv("D:/Python Dataset/train.csv") #Reading the dataset in a dataframe using Pandas
df.head(10)                                         #data exploration
df.describe()                                       #Summary numerical fields
df['Property_Area'].value_counts()
df['ApplicantIncome'].hist(bins=50)                 #distribution analysis
df['LoanAmount'].hist(bins=10)
#Categorical variable analysis
print 'Frequency Table for Credit History:'
temp1 = df['Credit_History'].value_counts(ascending=True)
print temp1
df.apply(lambda x: sum(x.isnull()),axis=0)          #find missing value in each column
df['LoanAmount'].fillna(df['LoanAmount'].mean(), inplace=True)  #Fill missing value in loan
amount
df.head(10)
df['Self_Employed'].value_counts()                  # Show Counts in Column with different
categories
df['Self_Employed'].fillna('No',inplace=True)       #fill  missing value with no
df.head(21)
#traet with extream values
df['LoanAmount_log']=np.log(df['LoanAmount'])
df['LoanAmount_log'].hist(bins=20)
```

SciPy

SciPy

Scipy (pronounced "Sigh Pie") is an open source Python library used by engineers doing scientific computing and technical computing.

Scipy contains modules for optimization, linear algebra, integration, interpolation, special functions, FFT, signal and image processing, ODE solvers and other tasks common in science and engineering.

Scipy builds on the NumPy array object and is part of the NumPy stack which includes tools like Matplotlib, pandas and SymPy.

This NumPy stack has similar users to other applications such as MATLAB and Scilab.

The NumPy stack is also sometimes referred to as the Scipy stack.

SciPy

Installation page:

http://www.scipy.org/install.html

http://sourceforge.net/projects/scipy/files/scipy/0.16.0b2/

SciPy

Sub-package Description	
cluster	Clustering algorithms
constants	Physical and mathematical constants
fftpack	Fast Fourier Transform routines
integrate	Integration and ordinary differential equation solvers
interpolate	Interpolation and smoothing splines
io	Input and Output
linalg	Linear algebra
ndimage	N-dimensional image processing
odr	Orthogonal distance regression
optimize	Optimization and root-finding routines
signal	Signal processing
sparse	Sparse matrices and associated routines
spatial	Spatial data structures and algorithms
special	Special functions
stats	**Statistical distributions and functions**
weave	C/C++ integration

Import Scipy

- The standard way of importing Scipy modules is:

  - **import scipy as sp**

- Scipy sub-packages need to be imported separately, for example

  - **from scipy import linalg, optimize**

  - **from scipy import stats**

Using scipy and scipy.stats

http://docs.scipy.org/doc/scipy/reference/stats.html

```
>>> from scipy import stats
>>> dir(stats)
['BinnedStatistic2dResult', 'BinnedStatisticResult', 'BinnedStatisticddResult', 'Tester', '__all__', '__builtins__', '__doc__', '__file
__', '__name__', '__package__', '__path__', '_binned_statistic', '_constants', '_continuous_distns', '_discrete_distns', '_distn_infras
tructure', '_distr_params', '_multivariate', '_rank', '_tukeylambda_stats', 'absolute_import', 'alpha', 'anderson', 'anderson_ksamp', '
anglit', 'ansari', 'arcsine', 'bartlett', 'bayes_mvs', 'bernoulli', 'beta', 'betai', 'betaprime', 'binned_statistic', 'binned_statistic
_2d', 'binned_statistic_dd', 'binom', 'binom_test', 'boltzmann', 'boxcox', 'boxcox_llf', 'boxcox_normmax', 'boxcox_normplot', 'bradford
', 'burr', 'callable', 'cauchy', 'chi', 'chi2', 'chi2_contingency', 'chisqprob', 'chisquare', 'circmean', 'circstd', 'circvar', 'combin
e_pvalues', 'contingency', 'cosine', 'cumfreq', 'describe', 'dgamma', 'dirichlet', 'distributions', 'division', 'dlaplace', 'dweibull',
 'entropy', 'erlang', 'expon', 'exponnorm', 'exponpow', 'exponweib', 'f', 'f_oneway', 'f_value', 'f_value_multivariate', 'f_value_wilks
_lambda', 'fastsort', 'fatiguelife', 'find_repeats', 'fisher_exact', 'fisk', 'fligner', 'foldcauchy', 'foldnorm', 'frechet_l', 'frechet
_r', 'friedmanchisquare', 'futil', 'gamma', 'gausshyper', 'gaussian_kde', 'genexpon', 'genextreme', 'gengamma', 'genhalflogistic', 'gen
logistic', 'gennorm', 'genpareto', 'geom', 'gilbrat', 'gmean', 'gompertz', 'gumbel_l', 'gumbel_r', 'halfcauchy', 'halfgennorm', 'halflo
gistic', 'halfnorm', 'histogram', 'histogram2', 'hmean', 'hypergeom', 'hypsecant', 'invgamma', 'invgauss', 'invweibull', 'invwishart',
'itemfreq', 'jarque_bera', 'johnsonsb', 'johnsonsu', 'kde', 'kendalltau', 'kruskal', 'ks_2samp', 'ksone', 'kstat', 'kstatvar', 'kstest'
, 'kstwobign', 'kurtosis', 'kurtosistest', 'laplace', 'levene', 'levy', 'levy_l', 'levy_stable', 'linregress', 'loggamma', 'logistic',
'loglaplace', 'lognorm', 'logser', 'lomax', 'mannwhitneyu', 'maxwell', 'median_test', 'mielke', 'mode', 'moment', 'mood', 'morestats',
'mstats', 'mstats_basic', 'mstats_extras', 'multivariate_normal', 'mvn', 'mvsdist', 'nakagami', 'namedtuple', 'nanmean', 'nanmedian', '
nanstd', 'nbinom', 'ncf', 'nct', 'ncx2', 'norm', 'normaltest', 'np', 'obrientransform', 'pareto', 'pdf_fromgamma', 'pearson3', 'pearson
r', 'percentileofscore', 'planck', 'pointbiserialr', 'poisson', 'power_divergence', 'powerlaw', 'powerlognorm', 'powernorm', 'ppcc_max'
, 'ppcc_plot', 'print_function', 'probplot', 'randint', 'rankdata', 'ranksums', 'rayleigh', 'rdist', 'recipinvgauss', 'reciprocal', 're
lfreq', 'rice', 'rv_continuous', 'rv_discrete', 's', 'scoreatpercentile', 'sem', 'semicircular', 'shapiro', 'sigmaclip', 'signaltonoise
', 'skellam', 'skew', 'skewtest', 'spearmanr', 'square_of_sums', 'ss', 'statlib', 'stats', 't', 'test', 'theilslopes', 'threshold', 'ti
ecorrect', 'tmax', 'tmean', 'tmin', 'triang', 'trim1', 'trim_mean', 'trimboth', 'truncexpon', 'truncnorm', 'tsem', 'tstd', 'ttest_1samp
', 'ttest_ind', 'ttest_ind_from_stats', 'ttest_rel', 'tukeylambda', 'tvar', 'uniform', 'variation', 'vonmises', 'vonmises_cython', 'von
mises_line', 'wald', 'warnings', 'weibull_max', 'weibull_min', 'wilcoxon', 'wishart', 'wrapcauchy', 'zipf', 'zmap', 'zscore']
>>>
```

Using scipy and scipy.stats

python

```
>>> import scipy as sp
>>> v = sp.randn(50)
>>> print V
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'V' is not defined
>>> print v
[  1.07738047e+00   9.43654303e-01   3.66712043e-01  -6.04381458e-01
  -1.02492583e+00  -2.03589980e+00   2.88630759e-01   4.81641402e-01
  -1.50367912e-01  -1.62357864e+00  -5.09008514e-01  -5.21631403e-01
  -4.69202374e-01  -7.03377555e-01   4.13467992e-01   7.83201849e-01
   1.83532362e+00  -1.55264062e+00   7.22062293e-02   1.52524535e+00
   3.66160916e-01   1.71988387e+00   4.52758776e-01  -3.41152879e-01
   1.01500933e-01   2.81863605e-01   8.63562302e-01   1.11998228e-01
   9.18018958e-04   4.22372737e-01   9.66390608e-02   1.28568261e+00
  -2.68864962e-01   1.36946801e+00  -7.48764392e-01  -2.28848317e-01
   4.90623659e-01   1.17655436e+00  -2.36660001e+00  -9.35248830e-01
   4.22107672e-01  -5.22794194e-01  -1.89990025e+00   9.79794256e-01
>>>
>>> v.mean()
0.021177758791554389
>>>
>>> v.min()
-2.3666000094081432
>>>
>>> v.var()
0.89123375311039621
>>>
>>> v.std()
0.94405177459204859
>>>
```

Using scipy and scipy.stats

```
>>> n, min_max, mean, var, skew, kurt = stats.describe(v)
>>> min_max[0]
-2.3666000094081432
>>> min_max[1]
1.835323622380758
>>> skew
-0.41780725742759545
>>> kurt
-0.033028602876918
>>>
```

Numerical Integration

The scipy.integrate sub-package provides several integration techniques including an ordinary differential equation integrator. An overview of the module is provided by the help command:

>>> **help(integrate)**

Methods for Integrating Functions given function object.

quad -- General purpose integration.

dblquad -- General purpose double integration.

tplquad -- General purpose triple integration.

fixed_quad -- Integrate func(x) using Gaussian quadrature of order n.

quadrature -- Integrate with given tolerance using Gaussian quadrature.

romberg -- Integrate func using Romberg integration.

Interface to numerical integrators of ODE systems.

odeint -- General integration of ordinary differential equations.

Bessel functions, first defined by the mathematician Daniel Bernoulli and then generalized by Friedrich Bessel, are the canonical solutions y(x) of Bessel's differential equation. In Python anonymous functions (without name)are defined using the lambda keyword.

Suppose you wish to integrate a bessel function jv(2.5,x) along the interval [0; 4.5]

$$I = \int_0^{4.5} J_{2.5}(x)\ dx.$$

```
import numpy as np
import scipy as sp
from scipy.integrate import quad
result = sp.integrate.quad(lambda x:sp.special.jv(2.5,x), 0, 4.5)
print result
```

Output:

(1.1178179380783244, 7.866317216380707e-09)

Linear Algebra

```
import numpy as np

import scipy as sp

A=np.mat('[1 10 7; 2 4 2; 9 2 8]')

B=np.mat('[12; 7; 5]')

A.I*B
```

Output:

```
matrix([[ 0.94318182],
        [ 1.71022727],
        [-0.86363636]])
```

OR

```
sp.linalg.solve(A,B)
```

Output:

```
array([[ 0.94318182],
       [ 1.71022727],
       [-0.86363636]])
```

→ Solving Linear equation

$$x + 10y + 7z = 12$$
$$2x + 4y + 2z = 7$$
$$9x + 2y + 8z = 5$$

$S = A^{-1} B$ where

S=[x y z] and B = [10 8 3]

Linear Algebra

→Finding Determinant

$$A = \begin{bmatrix} 135 \\ 251 \\ 238 \end{bmatrix}$$

$$|A| = 1\begin{vmatrix} 51 \\ 38 \end{vmatrix} - 3\begin{vmatrix} 21 \\ 28 \end{vmatrix} + 5\begin{vmatrix} 25 \\ 23 \end{vmatrix}$$

1(5.8-3.1)-3(2.8-2.1)+5(2.3-2.5)=-25

```
In [25]: C = np.mat('[1 3 5; 2 5 1; 2 3 8]')

In [26]: sp.linalg.det(C)
Out[26]: -25.000000000000004
```

$$I=\sqrt{\frac{2}{\pi}}\left(\frac{18}{27}\sqrt{2}\cos\left(4.5\right)-\frac{4}{27}\sqrt{2}\sin\left(4.5\right)+\sqrt{2\pi}\mathrm{Si}\left(\frac{3}{\sqrt{\pi}}\right)\right)$$

```
import numpy as np
import scipy as sp
I=np.sqrt(2/np.pi)*(18/27*np.sqrt(2)*np.cos(4.5)-
4/27*np.sqrt(2)*np.sin(4.5)+np.sqrt(2*np.pi)*sp.special.fresnel(3/np.sqrt(np.pi))[0])
print I
```

Output:

(1.11781793809+0j)

Reference links

http://nbviewer.jupyter.org/urls/bitbucket.org/hrojas/learn-pandas/raw/master/lessons/07%20-%20Lesson.ipynb

http://pandas.pydata.org/pandas-docs/stable/10min.html

Aegis
SCHOOL OF BUSINESS
SCHOOL OF DATA SCIENCE
SCHOOL OF TELECOMMUNICATION

Dropping entries from an axis

- On series or DataFrame, drop a row by his index

```
In: s
Out:
a      3.0
b      7.0
c      4.0
d      4.0
d      0.3
```

```
In: s.drop('d')
Out:
a      3.0
b      7.0
c      4.0
```

```
In: s.drop_duplicates()
Out:
a      3.0
b      7.0
c      4.0
d      0.3
```

- In DataFrame, (default) 'axis=0' refers to (row) index and axis=1 to columns

```
In: df
Out:
   Paris  Berlin  Madrid
b      0       1       2
a      3       7       5
c      6       4       8
```

```
In: df.drop('c')
Out:
   Paris  Berlin  Madrid
b      0       1       2
a      3       7       5
```

```
In :df.drop('Berlin', axis=1)
Out:
   Paris  Madrid
b      0       2
a      3       5
c      6       8
```

Exercise

Drop rows containing `'Rank' = 0`