In [1]:
```
# Import the pandas library for managing data
#
import pandas as pd
```

In [2]:
```
# Set the notebook so that it can display all countries in a dataframe
#
pd.set_option('display.max_rows', 200)
```

In [3]:
```
# Define variables for reading in the data worksheet from the World Ban
k's Doing Business website
#
data_url = 'http://www.doingbusiness.org/~/media/WBG/DoingBusiness/Docum
ents/Data/DB18-Historical-data-complete-data-with-DTFs.xlsx'
sheet = "All Data"
header_row = 1 # the convention in Python is 0-based indexes; in the wor
ksheet the header is in row 2
#
```

In [4]:
```
# Read the data sheet into a pandas dataframe df
# Use the header to get variable / column names
# Convert the value 'No Practice' to NaN - not a number
#
df = pd.read_excel(data_url,sheet_name = sheet, header = header_row, na_
values = ['No Practice'])
```

In [5]:
```
# Remove the #-delimiter on the line below, to see rows that include Ban
gladesh
#
#df[150:200]
#
# It is one of the countries where a second city was added. It is becaus
e of these additions
#    that later there is a cell that drops rows if
#
#       "len(df2.loc[i,'code']) != 3"
#
#    that is, if the code has more or less characters than the 3-letters
 that were used for the original observations
#
```

In [6]:
```
# Create a dictionary with keys that are the existing variable names and
 values that are the new names I use
#
# Note that the import from the excel sheet includes some random line br
eak characters '\n'
#
rename_variables = {'Country code': 'code',
                    'DB Year': 'year',
                    'Procedures - Men (number) ': 's_procs',
                    'Time - Men (days)': 's_time',
                    'Cost - Men (% of income per capita)': 's_cost',
                    'Minimum capital (% of income per capita)': 's_min_c
ap',
                    'Procedures (number)': 'cn_procs', 'Time (days)': 'c
n_time',
                    'Cost (% of Warehouse value)': 'cn_cost',
                    'Procedures (number).1': 'e_procs',
                    'Time (days).1': 'e_time',
                    'Cost (% of income per capita)': 'e_cost',
                    'Procedures (number).2': 'rp_procs',
                    'Time (days).2': 'rp_time',
                    'Cost (% of property value)': 'rp_cost',
                    'Strength of legal rights index (0-12) (DB15-18 meth
odology) ': 'ct_s',
                    'Depth of credit information index (0-8) (DB15-18 me
thodology) ': 'ct_d',
                    'Extent of conflict of interest regulation index (0-
10)\n(DB15-18 methodology) ':'pm_cft',
                    'Extent of shareholder governance index (0-10) (DB15
-18 methodology) ':'pm_gv',
                    'Payments (number per year)': 't_p', 'Time (hours pe
r year)':
                    't_t', 'Total tax rate (% of profit)': 't_tr',
                    'Time (days).3': 'en_time',
                    'Cost (% of claim)': 'en_cost',
                    'Recovery rate (cents on the dollar)': 'ri_r',
                    'Strength of insolvency framework index (0-16) (DB15
-18 methodology)': 'ri_s'
                   }

# Treat the variable names as a set so that set subtraction specifies th
e one to drop
#
all_vars = set(df.columns.values)

# Old_names from for the variables that the code keeps and renames
old_names = set(rename_variables.keys())

# The implied set of variables to drop
vars_to_drop = all_vars - old_names

# Create a list with the new names for the variables that remain
new_names = list(rename_variables.values())
```

In [7]:
```
# Make an independent copy of the dataframe.
#
# If I want to redo my calculations as I work interactively, I can just
 re-execute this cell without
#     reloading the data from the external website
#
df2 = pd.DataFrame(df.copy(deep = True))
```

In [8]:
```
# Convert the set of variables that I will drop from a set to a list
#
el_2 = list(vars_to_drop)
#
#len(df2.columns.values) # count of variables before the drop
#
# Drop the variables
#
df2.drop(el_2, axis = 1, inplace = True)
#
#len(df2.columns.values) # to check the variable count after the drop
```

In [9]:
```
# Rename the columns / variables that remain
#
df2.rename(columns=rename_variables, inplace = True)
#
# To verify that rename and drop affects df2 but not df, uncomment one,
 then other of next two lines and compare
#df
#df2
```

In [10]:
```
# Drop years before and including 2013
#
df2.drop(labels = [i for i in df2.index if df2.loc[i,'year'] <= 2013], i
nplace = True)
#
#len(df2)
```

In [11]:
```
# As noted above, drop recently added extra cities
#
df2.drop(labels = [i for i in df2.index if len(df2.loc[i,'code']) != 3],
 inplace = True)
#
#len(df2)
```

In [12]:
```
# Specify a multi-index for the remaining variables
#
df2.set_index(['year', 'code'], inplace=True)
#
#df2 # inspect results
```

In [13]:
```
# Create a separate dataframe to store the normalized or distance to the
 frontier (DTF) values
#    for different indicators.
#
dtf = pd.DataFrame(df2.copy(deep = True))
```

In [14]:
```
# Create a list of all the indicators that remain

el_3 = list(dtf.columns.values)
#el_3
#
#len(el_3)
```

In [15]:
```
# Indicators in high are ones where bigger values are better; opposite f
or variables in low
#
high = ['ct_s', 'ct_d', 'pm_cft', 'pm_gv',  'ri_r', 'ri_s' ]
low = ['s_procs', 's_time', 's_cost', 's_min_cap', 'cn_procs', 'cn_time'
,
       'cn_cost', 'e_procs', 'e_time', 'e_cost', 'rp_procs', 'rp_time',

       'rp_cost', 't_p', 't_t', 't_tr', 'en_time', 'en_cost']
#
# len(high) + len(low)
```

In [16]:
```
# This loop calculates the distance to the frontier for the 24 variables
 that are available in a consistent
#    form for the years I consider, DB2014-18, or calendar 2013-17.
#
# These normalized values are stored in the dtf dataframe. The raw value
s remainin in the df2 dataframe.
#
# The loop defines the distance to the frontier by taking the biggest an
d smallest values for
#    each variable in any year from DB years 2014-18.
#
# I wrote the code as I did assuming that I would use the max and min in
 each year; then I found
#    that they change over time, sometimes substantially. This is the ty
pe of issue that I understand only
#    if I work directly with the data myself.
#
# This problem is that the min (worst) value for an indicator can change
 dramatically based on
#    what happens in a single country with a very bad business environme
nt.
#    So I added the two lines that calculate mn_m and mx_m by taking the
 min and
#    max over all DB years from 2014 to 2018. This decision influences t
he relative influence that
#    different indicators have in my results.
#
# This is an important point. Suppose that it takes every other country
 between 10 and 100 days to
#    to issue a permit, but in one laggard it takes 10,000 days. Then al
```

```
l other countries will have a DTF
#    score for this indicator in the range 10/10,000 to 100/10,000. In t
his case, a country that takes only
#    10 days gets almost no recognition for its better performance relat
ive to a country that takes 100.
#
# The DTF value for this permit indicator will be 0.999 for the country
 that takes 10 days
#    and 0.990 for the country that takes 100 days. When this indicator
 is averaged along with 8 or 9 others,
#    it will have an effect on the overall indicator that is visible onl
y in the third decimal place. It will
#    be swamped by variation in other indicators.
#
# My choice is not one that I would defend as being the right way to det
ermine the relative influence of
#    different indicators. It underweights indicators with a fat lower
 (that is worse) tail. I haven't explored
#    the sensitivity of the results for Chile to alternative choices bec
ause I didn't want to be accused of
#    manipulating the data to get some particular outcome.
#
# For my purpose, the choice I made had the advantage that it is arbitra
ry and leads to rankings for
#    countries that do not change from year to year because of year to y
ear **changes** in the min (worst)
#    value of an indicator in some lagging country. My choice ensures th
at he range from best to worst,
#    and hence the relative influence of each indicator, stays fixed ove
r all the years that I consider.
#
# One of the advantages of making this code available is that it lets ot
hers do their own sensitivity analysis
#    with respect to this or any other issue.
#
# The approach used by the Doing Business team addresses this concern in
 a different way.
#    For most indicators (but not all), they also take the min and max o
ver a five year interval.
#    See the description of their approach here:
#
#     http://www.doingbusiness.org/~/media/WBG/DoingBusiness/Documents/A
nnual-Reports/English/DB18-Chapters/DB18-DTF-and-DBRankings.pdf
#
# To view the values for the min and max for each variable over all year
s or year by year, uncomment the
#    print statements in this loop.
#
#

for i in range(len(el_3)):
    mn = df2.groupby(['year'])[el_3[i]].min()
    #print ('mn = ', mn)
    mn_m = mn.min()
    #print('mn_m ',mn_m)
    mx = df2.groupby(['year'])[el_3[i]].max()
    #print ('mx = ', mx)
```

```
    mx_m = mx.max()
    #print('mx_m ',mx_m)
    if el_3[i] in low:
        dtf[el_3[i]] = (mx_m - df2[el_3[i]]) / (mx_m - mn_m)
    else:
        dtf[el_3[i]] = (df2[el_3[i]] - mn_m)  / (mx_m - mn_m)
```

In [17]:
```
#df2[4:5]  # Test raw data for visual comparison with Bank numbers from
 spreadsheet
```

In [18]:
```
#dtf[4:5]  # Test dtf or normalized data for comparison with Bank number
s on Afghanistan
```

In [19]:
```
# Follow the Bank's hierarchical procedure; average the sub-components o
f the different indicators
#
# d_ => prefix that means "distance to ..."
# s => Starting A Business ...
# cn => Construction Permits
# e => Getting Electricity
# rp => Registering Property
# ct => Contract Enforcement
# pm => Protection for Minority investors
# t => Taxes
# en => Enforcing Contracts
# ri => Resolving Insolvencies
d_s = pd.Series((dtf['s_procs'] + dtf['s_time'] + dtf['s_cost'] + dtf['s
_min_cap']) / 4)
d_cn = pd.Series((dtf['cn_procs'] + dtf['cn_time'] +  dtf['cn_cost']) /
3)
d_e = pd.Series((dtf['e_procs'] + dtf['e_time'] + dtf['e_cost']) / 3)
d_rp = pd.Series((dtf['rp_procs'] + dtf['rp_time'] + dtf['rp_cost']) / 3
)
d_ct = pd.Series((dtf['ct_s'] + dtf['ct_d']) / 2)
d_pm = pd.Series((dtf['pm_cft'] + dtf['pm_gv']) / 2)
d_t = pd.Series((dtf['t_p'] + dtf['t_t'] + dtf['t_tr']) / 3)
d_en = pd.Series((dtf['en_time'] + dtf['en_cost']) / 2)
d_ri = pd.Series((dtf['ri_r'] + dtf['ri_s']) / 2)
```

In [20]:
```
# The overall average across indicators
# I have 9 here because none of the indicators of trade costs are availa
ble for all 5 years
# It should be easy to tweak the code to include some of the trade indic
ators but
#     at the cost of restricting the analysis to the 4 data years 2014 t
o 2017
#
d_DTF = pd.Series((d_s + d_cn + d_e + d_rp + d_ct + d_pm + d_t + d_en +
d_ri) / 9)
#d_DTF
```

In [21]:
```
df2 = pd.concat([df2, d_s.rename('s'),
                 d_cn.rename('cn'),
                 d_e.rename('e'),
                 d_rp.rename('rp'),
                 d_ct.rename('ct'),
                 d_pm.rename('pm'),
                 d_t.rename('t'),
                 d_en.rename('en'),
                 d_ri.rename('ri'),
                 d_DTF.rename('DTF')], axis=1)
#df2
#len(df2)
```

In [22]:
```
df2.dropna(axis = 0, subset = ['DTF'], inplace=True)
```

In [23]:
```
#df2
#len(df2)
```

In [24]:
```
# Define 5 series, one for each year, indexed by country code
#
d_2018 = pd.Series(df2.loc[2018]['DTF'])
#len(d_2018)
d_2017 = pd.Series(df2.loc[2017]['DTF'])
#len(d_2017)
d_2016 = pd.Series(df2.loc[2016]['DTF'])
#len(d_2016)
d_2015 = pd.Series(df2.loc[2015]['DTF'])
#len(d_2015)
d_2014 = pd.Series(df2.loc[2014]['DTF'])
#len(d_2014)
#
```

In [25]:
```
# Create series objects that I can sort, one for each year
#
df_2018 = pd.DataFrame(d_2018)
dfs_2018 = df_2018.sort_values(by =['DTF'], axis=0, ascending=False, inp
lace=False, kind='quicksort')
length=len(dfs_2018)
dfs_2018['Rank18']= pd.Series(range(1, length + 1 ,1), index=dfs_2018.in
dex)

df_2017 = pd.DataFrame(d_2017)
dfs_2017 = df_2017.sort_values(by =['DTF'], axis=0, ascending=False, inp
lace=False, kind='quicksort')
length=len(dfs_2017)
dfs_2017['Rank17']= pd.Series(range(1, length + 1 ,1), index=dfs_2017.in
dex)

df_2016 = pd.DataFrame(d_2016)
dfs_2016 = df_2016.sort_values(by =['DTF'], axis=0, ascending=False, inp
lace=False, kind='quicksort')
length=len(dfs_2016)
dfs_2016['Rank16']= pd.Series(range(1, length + 1 ,1), index=dfs_2016.in
dex)

df_2015 = pd.DataFrame(d_2015)
dfs_2015 = df_2015.sort_values(by =['DTF'], axis=0, ascending=False, inp
lace=False, kind='quicksort')
length=len(dfs_2015)
dfs_2015['Rank15']= pd.Series(range(1, length + 1 ,1), index=dfs_2015.in
dex)

df_2014 = pd.DataFrame(d_2014)
dfs_2014 = df_2014.sort_values(by =['DTF'], axis=0, ascending=False, inp
lace=False, kind='quicksort')
length=len(dfs_2014)
dfs_2014['Rank14']= pd.Series(range(1, length + 1 ,1), index=dfs_2014.in
dex)
```

In [26]:
```
# Create a new dataframe object dfs indexed by country code that holds s
orted series
#
# Combine the sorted series into a single dataframe indexed by country
#
dfs_all = pd.concat([dfs_2014, dfs_2015, dfs_2016, dfs_2017,dfs_2018], a
xis=1)
```