```
import pandas as pd
from operator import itemgetter
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
```

```
# Collecting Stock data of China and Visulaizing Data frame
data=pd.read_csv("drive/My Drive/ECO764_group_Assn1/CE_China.csv")
df=data.drop(data.columns[1:7],axis=1)
# Exctrating data between 2013-2018
df=df.iloc[99:171,:]
df
```

	Name	INDUSTRIAL AND COMM - MARKET CAPITALIZATION	INDUSTRIAL AND COMM - RETURN ON EQUITY - TOTAL (%)	INDUSTRIAL AND COMM - PRICE INDEX	KWEICHOW MOUTAI CO - MARKET CAPITALIZATION	KWEICHO MOUTA CO RETUR O EQUITY TOTA (%
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	.4
100	02/28/2013	1.257971e+09	21.91	130.6	133281543.0	39.4
101	03/28/2013	1.257971e+09	21.91	124.4	133281543.0	39.4
102	04/28/2013	1.257971e+09	21.91	125.7	133281543.0	39.4
103	05/28/2013	1.257971e+09	21.91	130.0	133281543.0	39.4
...	...	...	...	...	...	.
166	08/28/2018	1.885389e+09	13.68	170.9	741169264.0	34.4
167	09/28/2018	1.885389e+09	13.68	179.0	741169264.0	34.4
168	10/28/2018	1.885389e+09	13.68	176.5	741169264.0	34.4
169	11/28/2018	1.885389e+09	13.68	166.0	741169264.0	34.4
170	12/28/2018	1.885389e+09	13.68	164.1	741169264.0	34.4

This file was updated remotely or in another tab. To force a save, overwriting the last update, select Save from the File menu

72 rows × 1018 columns

```
import numpy as np
import math
from google.colab import files
def avg(list):
  list=np.array(list)
  return np.sum(list)/len(list)
def intersection(lst1, lst2):
    lst3 = [value for value in lst1 if value in lst2]
    return lst3
def isNaN(x):
    return str(float(x)).lower() == 'nan'

# get_smb_hml returns the SMB and HML of the given month represented by a row
def get_smb_hml(df,row):
  total=0
  cnt=0
  m_cap_list=[]
  roe_list=[]
  return_list=[]
  idx=0
  z=df.iloc[row]
  z1=df.iloc[row-1]
  for i in range(int((len(df.columns)-1)/3)):
    total=total+1
    m_cap=z[3*i+1]
    roe=z[3*i+2]
    price_t=z[3*i+3]
    price_t_1=z1[3*i+3]
    if not isNaN(m_cap) and not isNaN(roe) and not isNaN(price_t) and not isNaN(pr
      m_cap_list.append((m_cap,idx))
      roe_list.append((roe,idx))
      ret=(math.log(price_t/price_t_1))*100
      return_list.append(ret)
```

This file was updated remotely or in another tab. To force a save, overwriting the last update, select Save from the File menu

```
  m_cap_list=sorted(m_cap_list,key=itemgetter(0))
  roe_list=sorted(roe_list,key=itemgetter(0))

# Dividing companies in SMALL/BIG and VALUE/NEUTRAL/GROWTH
  small=m_cap_list[0:int(len(m_cap_list)/2)]
  small_i=[b for (a,b) in small]
  big=m_cap_list[int(len(m_cap_list)/2):int(len(m_cap_list))]
  big_i=[b for (a,b) in big]
  growth=roe_list[0:int(len(roe_list)*0.3)]
  growth_i=[b for (a,b) in growth]
  neutral=roe_list[int(len(roe_list)*0.3):int(len(roe_list)*0.7)]
  neutral_i=[b for (a,b) in neutral]
  value=roe_list[int(len(roe_list)*0.7):int(len(roe_list))]
  value_i=[b for (a,b) in value]

# Created 6 posrtfolios SV, SN, SG and BN, BG, BV
  sv=intersection(small_i,value_i)
  sn=intersection(small_i,neutral_i)
  sg=intersection(small_i,growth_i)
  bv=intersection(big_i,value_i)
  bn=intersection(big_i,neutral_i)
  bg=intersection(big_i,growth_i)

  sv_return_list=[return_list[i] for i in sv]
  sn_return_list=[return_list[i] for i in sn]
  sg_return_list=[return_list[i] for i in sg]
  bv_return_list=[return_list[i] for i in bv]
  bn_return_list=[return_list[i] for i in bn]
  bg_return_list=[return_list[i] for i in bg]

  # Calculating AVG return of the Portfolios
  sv_return=avg(sv_return_list)
  sn_return=avg(sn_return_list)
  sg_return=avg(sg_return_list)
  bv_return=avg(bv_return_list)
  bn_return=avg(bn_return_list)
  bg_return=avg(bg_return_list)

# Calculating SML and HML
  sml=0.33*(sv_return+sn_return+sg_return)-0.33*(bv_return+bn_return+bg_return)
  hml=0.5*(sv_return+bv_return)-0.5*(sg_return+bg_return)
  return [sml,hml]



data=[]
# Calculating SML and HML and Return_of_INDUSTRIAL_and_COMM for all months
for i in range(1,72):
  x=get_smb_hml(df,i)
  x.insert(0,df.iloc[i,0])
  price_t=df.iloc[i,3]
  price_t_1=df.iloc[i-1,3]
```

This file was updated remotely or in another tab. To force a save, overwriting the last update, select Save from the File menu

```
  data.append(x)

df1 = pd.DataFrame(data, columns = ['Date', 'SMB','HML','Return_of_INDUSTRIAL AND
df1.head(10)
```

```
         Date        SMB        HML  Return_of_INDUSTRIAL AND COMM
0  02/28/2013   1.265420   1.311712                      -1.896151
1  03/28/2013   0.611300   4.883151                      -4.863704
2  04/28/2013  -1.445221   3.404877                       1.039594
3  05/28/2013   4.854283   0.784893                       3.363633
4  06/28/2013  -1.284889   9.406113                      -4.162360
5  07/28/2013   4.288953   0.603131                      -2.764404
6  08/28/2013   4.895836   1.072659                      -0.744728
7  09/28/2013  -1.437084  -1.436405                      -0.499585
8  10/28/2013   1.405488   3.120159                      -2.621715
9  11/28/2013  -1.120437  -3.850613                       1.023027
```

```
# Exporting into respective CSV files
df1.to_csv('regression_data.csv')
files.download('regression_data.csv')

# Function to get Momentum Factor for a month
def get_momentum_factor(df,row):
  total=0
  cnt=0
  m_cap_list=[]
  return_list=[]
  return11_list=[]
  idx=0
  z=df.iloc[row]
  z1=df.iloc[row-1]
  tm1=df.iloc[row-1]
  tm12=df.iloc[row-12]

  for i in range(int((len(df.columns)-1)/3)):
    total=total+1
    m_cap=z[3*i+1]
    price_t=z[3*i+3]
    price_t_1=z1[3*i+3]
    price_tm1=tm1[3*i+3]
    price_tm12=tm12[3*i+3]

    if not isNaN(m_cap) and not isNaN(price_tm1) and not isNaN(price_tm12) and not
      m_cap_list.append((m_cap,idx))
```

This file was updated remotely or in another tab. To force a save, overwriting the last update, select Save from the File menu

```
      ret_11=(math.log(price_tm1/price_tm12))*100
      return11_list.append((ret_11,idx))
      idx=idx+1
      cnt=cnt+1

  m_cap_list=sorted(m_cap_list,key=itemgetter(0))
  momentum_return=sorted(return11_list,key=itemgetter(0))

  # Finding Winner and Loser portfolios
  loser=momentum_return[0:int(len(momentum_return)*0.3)]
  winner=momentum_return[int(len(momentum_return)*0.7):int(len(momentum_return))]

  small=m_cap_list[0:int(len(m_cap_list)/2)]
  small_i=[b for (a,b) in small]
  big=m_cap_list[int(len(m_cap_list)/2):int(len(m_cap_list))]
  big_i=[b for (a,b) in big]

  winner_index=[b for (a,b) in winner]
  loser_index=[b for (a,b) in loser]

# Forming Winner-big, Winner-small, Loser-big, Loser-small portfolios
  WB=intersection(big_i,winner_index)
  WS=intersection(small_i,winner_index)
  LB=intersection(big_i,loser_index)
  LS=intersection(small_i,loser_index)
```

```
# Finding equally weigted returns
  WS_VALUE=[a for (a,b) in return_list if b in WS]
  WB_VALUE=[a for (a,b) in return_list if b in WB]
  LS_VALUE=[a for (a,b) in return_list if b in LS]
  LB_VALUE=[a for (a,b) in return_list if b in LB]

  WS_VALUE=avg(WS_VALUE)
  WB_VALUE=avg(WB_VALUE)
  LS_VALUE=avg(LS_VALUE)
  LB_VALUE=avg(LB_VALUE)


  return (WS_VALUE-LS_VALUE+WB_VALUE-LB_VALUE)/2;


data=[]

for i in range(13,72):
  # print(i)
  x=[get_momentum_factor(df,i)]
  x.insert(0,df.iloc[i,0])
  # print(x)
  data.append(x)

# print(data)
df2 = pd.DataFrame(data, columns = ['Date', 'Momentum_Factor'])
```

This file was updated remotely or in another tab. To force a save, overwriting the last update, select Save from the File menu

```
df2.head(10)
```

```
         Date  Momentum_Factor
0  02/28/2014         2.554836
1  03/28/2014        -5.608138
2  04/28/2014        -1.955867
3  05/28/2014         0.727727
4  06/28/2014         1.842021
5  07/28/2014        -5.157732
6  08/28/2014         0.147788
7  09/28/2014        -0.021563
8  10/28/2014         1.491300
9  11/28/2014        -5.076985
```

This file was updated remotely or in another tab. To force a save, overwriting the last update, select Save from the File menu