**DataViva**

Processo de Cálculo

V0.1

Núcleo de Sistemas e Gestão

Setembro 2013

Histórico de Revisão

|  |  |  |  |
| --- | --- | --- | --- |
| DATA | VERSÃO | DESCRIÇÃO | AUTOR |
| 29/07/2011 | 1.0 | Criação do documento | Mario H. Mol |
|  |  |  |  |

ÍNDICE

[1. INTRODUÇÃO 5](#_Toc372723477)

[2. ARQUITETURA 6](#_Toc372723478)

[2.1. Dependências 6](#_Toc372723479)

[2.2. Growth Lib 7](#_Toc372723480)

[2.3. Configuração 7](#_Toc372723481)

[2.4. Execução 8](#_Toc372723482)

[3. PROCESSOS 9](#_Toc372723483)

[3.1. Agregação 9](#_Toc372723484)

[3.2. Desagregação 10](#_Toc372723485)

[3.3. Taxa e Valor de Crescimento 11](#_Toc372723486)

[3.1. Diversidade 12](#_Toc372723487)

[3.2. Funções 13](#_Toc372723488)

[3.2.1. NumPy 14](#_Toc372723489)

[3.2.2. Pandas 15](#_Toc372723490)

[4. RAIS 18](#_Toc372723491)

[4.1. Indicador Recomendado 19](#_Toc372723492)

[4.2. Importância 19](#_Toc372723493)

[4.3. RCA, Distribuição e Oportunidade 20](#_Toc372723494)

[5. SECEX 22](#_Toc372723495)

[5.1. Indicadores de Complexidade: Observatory 23](#_Toc372723496)

[5.2. Indicadores de Complexidade Econômica 23](#_Toc372723497)

[5.1. RCA, Distribuição e Oportunidade 24](#_Toc372723498)

[6. HOMOLOGAÇÃO 26](#_Toc372723499)

[6.1. Agregações 26](#_Toc372723500)

[6.2. Localidades 27](#_Toc372723501)

[6.3. Indústrias 28](#_Toc372723502)

[6.4. Ocupações 29](#_Toc372723503)

[6.5. Banco de Dados 30](#_Toc372723504)

[6.6. RAIS 30](#_Toc372723505)

[6.6.1. ISIC 31](#_Toc372723506)

[6.7. SECEX 32](#_Toc372723507)

INTRODUÇÃO

Está sendo implantado em Minas Gerais o *Project Space* (PS), em parceria com a *Growth Ventures* (GV) e o Governo do Estado de Minas Gerais. Criado pelos professores Ricardo Hausmann (Universidade de Harvard) e Cesar Hidalgo (Instituto de Tecnologia de Massachusetts), no ano de 2007, o Product Space é um conjunto de ferramentas, conceitos e metodologia que tenta explicar o desenvolvimento desigual de estados ou regiões e criar instrumentos para que esse processo seja feito de maneira mais uniforme.

Uma entrega deste projeto é uma plataforma Web que disponibilizará para seus usuários uma ferramenta de pesquisa e análise de um grande volume de dados, utilizando a vanguarda da exibição de dados com gráficos de última geração. Os desenvolvedores da GV entregaram ao Escritório de Prioridades Estratégicas de Minas Gerais (EPE-MG) os fontes deste sistema e conhecimento para que seja possível dar continuidade na sua manutenção e evolução. Utilizando os fontes foi realizada uma análise para garantir que, novos desenvolvedores possam trabalhar neste código da melhor forma possível.

Esta análise do sistema tem por objetivo principal orientar o desenvolvedor que for trabalhar na evolução deste projeto, resultando neste documento. Sendo assim temos aqui um documento a ser utilizado como guia para a manutenção ou desenvolvimento de todo o processo de cálculo envolvido para a criação deste projeto.

ARQUITETURA

Junto com o código fonte existe uma pasta chamada scripts com todos os arquivos necessários para realização do processo de cálculo. Esta parte de cálculo está em uma branch separada da máster, chamada scripts, e pode ser encontrada no endereço:

* <https://github.com/alexandersimoes/dataviva/tree/scripts>

Veja a estrutura de pastas completa:

* \ : Na raiz do diretório temos um arquivo com as tecnologias que o projeto depende para ser executado, além de dois arquivos de comando para executar todo o processo de cálculo
* \scripts\: Nesta pasta temos os arquivos de criação deste módulo de cálculo (config, init e helpers), além das pastas abaixo
* \scripts\rais : Arquivos com os passos de 1 a 7 referente a execução da Rais
* \scripts\secex: Arquivos com passos de 1 a 8 para os cálculos da Secex.
  1. Dependências

Existem bibliotecas específicas do Python que precisam ser instaladas para executar este processo. Vamos começar por aquelas que necessita de instalação manual, inclusive em caso de máquinas Windows. A primeira é a NumPy:

* <http://sourceforge.net/projects/numpy/files/NumPy/>

Utilizar a seguinte versão:

* <http://sourceforge.net/projects/numpy/files/NumPy/1.7.1/numpy-1.7.1-win32-superpack-python2.7.exe/download>

Para conferir, veja se pelo Pip mostra como instalado:

* pip install numpy

A biblioteca de cálculo matemático Panda tem na versão de 32 Bits e na versão 64 Bits, confira após a instalação com o pip para garantir que foi instalado:

* http://pandas.pydata.org/
  + pandas-0.12.0.win32-py2.7.exe
  + pandas-0.12.0.win-amd64-py2.7.exe

Nas instalações seguintes podemos seguir instalando através do Pip, por meio dos comandos:

* pip install patsy
* pip install python-dateutil
  1. Growth Lib

Para instalar o Growth Lib é necessário baixar os fontes da pagina do Alex Simões na rede GitHUb, através do endereço:

* https://github.com/alexandersimoes/growth

Existem bibliotecas específicas do Python que precisam ser instaladas para executar este processo:

* numpy , matplotlib
* scipy, scikits.statsmodels
* pandas , patsy

Além da biblioteca arquivos de dados já calculados são utilizados também deste projeto, o *Atlas Observatory*, vide mais informações em:

* http://atlas.media.mit.edu/
  1. Configuração

Algumas variáveis de ambiente são necessárias para a execução deste processo, são elas:

* DATA\_DIR: Aponta para o diretório onde está as bases originais e onde as informações calculadas serão guardadas.
  + Este diretório precisa ter pastas com nome rais e secex, correspondendo ao arquivo texto original.
* DATAVIVA\_DB\_USER, DATAVIVA\_DB\_PW e DATAVIVA\_DB\_NAME
  + Informações relativas a configuração com o banco de dados

Existe a seguinte convenção de nome para estes arquivos de dados originais:

* Rais: Rais<year>.csv[.zip, .gzip, .bz2]
* Secex: MDIC\_<year>.csv[.zip, .gzip, .bz2]

Estes arquivos podem estar comprimidos, nos formatos zip, gzip ou bzip2, ou de forma crua, no formato .csv. Como saída é esperado uma tabela com todas as agregações possíveis e outras derivadas delas com as combinações de agregações. A tabela completa é chamada de YBIO, isto porque contém as seguintes agregações:

* Y – Year – Ano
* B – Localidade
* I – Industry – Atividade Econômica
* O – Occupation - Ocupação

Tomando como ideia estas agregações, são criadas então as seguintes tabelas:

* YBO, YBI, YIO
* YO, YI, YB

Para exemplificar o uso destas tabelas na aplicação, vamos dizer que precisamos saber as ocupações por indústria, para este caso a informação estaria disponível na YIO.

* 1. Execução

A execução pode ser realizada passo a passo ou completa. Para executar um passo em específico existe a possibilidade de executar via python, como no exemplo abaixo:

* python -m scripts.rais.step\_1\_aggregate -y <year>
  + Exemplo de execução do passo 1 para a base rais
* python -m scripts.rais.step\_1\_aggregate -y <year>
  + Exemplo de execução do passo 1 para a base secex

Nos exemplos foi utilizado a opção –m , que diz para que seja executado como um módulo.

Em caso de realizar a execução completa, temos um arquivo de execução em lotes, chamado de bash em máquinas Linux. Neste script ele faz a chamada de execução do python para cada passo existente, como no exemplo:

* bash rais\_full\_year.sh <year>
  + Exemplo de execução complete em bash para a Rais
* [SECEX] bash secex\_full\_year.sh <year>
  + Exemplo de execução complete em bash para a Secex

Por fim, quando teremos todos os valores de cálculo realizado, temos um script para a importação destes dados no banco de dados MySQL, que passara então a ser lido pela aplicação e finalizando assim o processo de execução dos cálculos.

1. PROCESSOS

Todo o processo de cálculo é bem parecido para as duas bases de dados, Secex e Rais. Sendo assim criamos este capítulo para documentarmos a implementação do processo como um todo. Assim nos capítulos seguintes vamos entrar no detalhe e especificidade de cada base de dados.

Para cada base de dados temos uma pasta em separado contendo:

* Arquivos *Python* para cada passo de cálculo, como por exemplo no caso da rais o primeiro passo é representado pelo arquivo step\_1\_aggregate.py
  + Estes sempre contêm um método main , ou seja, a função inicial a ser executado para iniciar o passo de cálculo
  + Além deste método devem existir outros métodos auxiliares para a execução do passo
* Para cada arquivo de *Python* de passo de cálculo, temos um correspondente em bash, para facilitar a execução em lote. Utilizando o exemplo acima, o arquivo para execução é o step\_1.sh
  + A ideia deste script é ter a execução de todos os anos com apenas uma chamada, um exemplo do conteúdo:
  + python scripts/rais/step\_1\_aggregate.py -d dataviva\_data/rais -y 2011
* Por fim temos um arquivo chamado import\_sql.sh, que contém um script bash para realizar a importação das tabelas criadas durante o processo para o banco de dados.

Veja que em todo momento é utilizado arquivos de execução bash, compatíveis apenas com sistema operacional Linux. Como utilizamos servidor Windows é necessário realizar uma transformação destes arquivos para arquivos do tipo Batch (.bat). Nos casos de execução de cada passo, não há uma necessidade de conversão, sendo necessário apenas para o arquivo de import\_sql.sh.

O resultado de cada etapa do processamento é gravado utilizando compactação BZ2, semelhante ao ZIP, tecnologia esta bem comum em ambientes Linux. Esta compactação é necessária, pois estamos trabalhando com um grande volume de dados e, para cada passo do processo de cálculo, os dados são praticamente duplicados.

* 1. Agregação

Este é o primeiro passo do processo de cálculo, isto é, dado um determinado ano o script faz a leitura do arquivo CSV, arquivo este que representa a entrada de dados de todo o processo. Esta base é a enviada pela equipe do Escritório, seja ela originada da Secex ou Rais, em formato texto (CSV). Para realizar a agregação é necessário chamar o script passando o ano de processamento, como no exemplo. Lembrando que para executar este passo é necessário informar o diretório o diretório onde os arquivos de base crua e os arquivos de processamento são gravados:

* python scripts/rais/step\_1\_aggregate.py -d /sites/dataviva\_data/rais -y 2003

Durante a leitura das informações é feito uma verificação no formato das classificações utilizadas, as seguintes funções realizam este tratamento:

* cbo\_format
* isic\_format
* munic\_format
* wage\_format

O propósito deste script é ler todos os dados e gravar um arquivo com todas agregações, portanto o YBIO.tsv. Durante este processo são calculados também informações desagregadas, conforme abaixo:

* cbo: 1 e 2 dígitos
* isic: 1 e 3 dígitos
* bra: 2 e 4 dígitos

O formato do arquivo de entrada é diferenciado para cada base, portanto vamos detalhar o formato esperado abaixo.

*Rais*: As colunas esperadas para o arquivo de entrada.

* 0: Year; 1: Employee\_ID; 2: Establishment\_ID; 3: Municipality\_ID;
* 4: BrazilianOcupation\_ID; 5: SBCLAS20; 6: CLASCNAE20; 7: WageReceived;
* 8: EconomicActivity\_ID\_ISIC; 9: Average\_monthly\_wage

*Secex*:

* 0: Year; 1: Municipality\_ID; 2: State\_ID
* 3: TransactedProduct\_ID\_HS; 4: DestinationCountry\_ID
* 5: TransactionAmount\_US$\_FOB;

O resultado final é gravado na pasta /secex/ybio.tsv.bz2 e /rais/ybio.tsv.bz2.

* 1. Desagregação

Segundo passo do processo de cálculo, realiza a criação das tabelas desagregadas baseada na tabela agregada, YBIO, portanto o arquivo /ybio.tsv.bz2, proveniente do primeiro passo do processo de cálculo. Para a execução deste script é necessário informar o ano de processamento, conforme abaixo:

* python step\_2\_disaggregate.py -f 2002/ybio.tsv -y 2002

Utilizando a YBIO são filtrados apenas in formações do nível mais desagregado para as classificações, isic\_id, cbo\_id e bra\_id. Para garantir este filtro, foi implementado da seguinte forma:

* if len(line["bra\_id"]) == 8 and len(line["isic\_id"]) == 5:
  + tables["yo"][line["cbo\_id"]]["wage"] += float(line["wage"])

As tabelas criadas tomando esta como origem são do formato TSV e compactadas com BZIP2. Para a RAIS as tabelas de saída são:

* YO, YI e YB
* YBO, YBI e YIO

Já no caso da SECEX as seguintes tabelas são criadas após a execução deste passo:

* YB YBW YBP
* YP YPW YW
  1. Taxa e Valor de Crescimento

Para este cálculo é necessário comparar a taxa de crescimento de um ano com o ano anterior e outro comparado com 5 anos atrás. Quando não existe um histórico para um determinado ano, a informação não é calculada. Esta taxa é criada para cada tabela desagregada (YB, YBI,YBO, YI, YIO e YO), e o mesmo é executado para a base da Rais e Secex.

No caso é feito o cálculo para o valor e a taxa de crescimento, portanto para valor é feito uma subtração e para taxa a divisão:

* print "calculating 1 year wage growth value"
* current["wage\_growth\_val"] = current["wage"] - prev["wage"]
* print "calculating 1 year wage growth rate"
* current["wage\_growth\_rate"] = (current["wage"] / prev["wage"]) - 1
* print "calculating 1 year num\_emp growth value"
* current["num\_emp\_growth\_val"] = current["num\_emp"] - prev["num\_emp"]
* print "calculating 1 year num\_emp growth rate"
* current["num\_emp\_growth\_rate"] = (current["num\_emp"] / prev["num\_emp"]) - 1

Este é realizado tanto para anual como para plurianual ( 5 anos):

* print "calculating 5 year wage growth value"
* current["wage\_growth\_val\_5"] = current["wage"] - prev\_5["wage"]
* print "calculating 5 year wage growth rate"
* current["wage\_growth\_rate\_5"] = (current["wage"] / prev\_5["wage"]) \*\* (1.0/5.0) - 1
* print "calculating 5 year num\_emp growth value"
* current["num\_emp\_growth\_val\_5"] = current["num\_emp"] - prev\_5["num\_emp"]
* print "calculating 5 year num\_emp growth rate"
* current["num\_emp\_growth\_rate\_5"] = (current["num\_emp"] / prev\_5["num\_emp"]) \*\* (1.0/5.0) – 1

Lembrando que para termos a data de crescimento, a seguinte fórmula é respeitada:

* Taxa de Crescimento =( ( presente / passado) \*\* (1/n) ) -1
  + Onde n é o valor de anos entre o presente e passado
  1. Diversidade

O cálculo da diversidade identifica o número de indústrias e ocupações únicas dado uma determinada localidade, seja cidade, regiões de planejamento ou meso regiões. Este é aplicado para as agregações YB, YI e YO. Portanto foi criado uma função para calcular utilizando como parâmetro o nome do arquivo desagregado. Vamos tomar por exemplo o YB, que utiliza de entrada o ybi.tsv e o ybo.tsv.

* yb\_isic\_diversity = get\_diversity(ybi\_file, "bra\_id", "isic\_id")
* yb\_cbo\_diversity = get\_diversity(ybo\_file, "bra\_id", "cbo\_id")

Desta informação, é realizado o filtro para obter os dados com o máximo de tamanho para as variáveis. Se tomarmos o exemplo do YB\_ISIC, teremos filtro para a atividade econômica com 4 dígitos, portanto a informação totalmente desagregada.

* criterion = tbl[column].map(lambda x: len(x) == get\_deepest(column))

Utilizando do pivot cria-se uma tabela utilizando na linha o bra\_id e na coluna o isic\_id, para os valores o numero de empregados. Realiza o truncamento para quando houver empregados ser igual a 1 e no caso oposto a 0.

* diversity = tbl.pivot(index=index, columns=column, values="num\_emp").fillna(0)

|  |  |  |
| --- | --- | --- |
|  | Mineração | Agricultura |
| BH | 1 | 1 |
| BETIM | 1 | 0 |

Por fim faz-se a soma de atividades econômicas que empregam alguém para uma determinada localidade:

* diversity = diversity.sum(axis=1)

|  |  |  |
| --- | --- | --- |
|  | BH | BETIM |
| Diversidade | 2 | 1 |

O mesmo cálculo é repetido para obter esta informação utilizando escala logarítmica., através do método:

* yb\_isic\_effective\_diversity = get\_effective\_diversity(ybi\_file, "bra\_id", "isic\_id")
  1. Funções

Passaremos pelas funções mais utilizadas durante o processo de cálculo, possibilitando assim uma análise mais profunda assim para ir a fundo durante o a análise do código.

*Sum*: Utilizando como entrada uma matriz, é relizado a soma dops valores encontrados na coluna. Portanto o resultado final é um vetor, com os valores totais encontrados para cada linha. A forma de uso desta função é conforme abaixo:

* denoms = rcas.sum()
  + rcas é uma matriz
  + denoms é um vetor com os valores somados

Exemplificando, uma matriz de Localidade x Indústria

|  |  |  |  |
| --- | --- | --- | --- |
|  | Mineração | Agricultura | Manufatura |
| BH | 1,001 | 0,33 | 1,2 |
| Betim | 0,56 | 0,33 | 1,23 |
| Contagem | 0,33 | 1,2 | 1,23 |

Aqui o exemplo do vetor encontrado após a aplicação de soma na matriz acima.

|  |  |  |
| --- | --- | --- |
| BH | Betim | Contagem |
| 2,531 | 2,12 | 2,76 |

*Drop*: Apenas remove as colunas, ou linhas, para diminuir o tamanho da matriz e aumentar o poder de processamento. Neste exemplo, foi escolhido o eixo 1 (axis=1) e então excluído as colunas year, num\_emp e num\_est:

* ybi.drop(["year", "num\_emp", "num\_est"], axis=1)

*Pivot*: Reconstrói uma tabela de dados utilizando valores nas colunas, em um formato onde temos eixo X e Y e no centro o valor. Como parâmetros temos:

* *index* : string ou object
  + Coluna de dados a ser utilizado como linha , ou index
* *columns* : string ou object
  + Coluna de dados a ser utilizada como coluna na nova tabela
* *values* : string ou object, optional
  + Coluna de dados a ser utilizado como o valor da nova tabela.

No exemplo abaixo temos uma tabela de dados com as colunas Foo, Bar e Baz:

|  |  |  |  |
| --- | --- | --- | --- |
| # | Foo | Bar | Baz |
| 0 | One | A | 1 |
| 1 | One | B | 2 |
| 2 | One | C | 3 |
| 3 | Two | A | 4 |
| 4 | Two | B | 5 |
| 5 | Two | C | 6 |

Se criarmos um pivot com a linha sendo o Foo, a coluna o Bar e o valor o Baz, teremos como resultado:

|  |  |  |  |
| --- | --- | --- | --- |
|  | A | B | C |
| One | 1 | 2 | 3 |
| Two | 4 | 5 | 6 |

Um exemplo de uso no Dataviva:

* ybi = ybi.pivot(index="bra\_id", columns="isic\_id", values="wage").fillna(0)
  + 1. NumPy

Existem funções específicas da biblioteca NumPy utilizadas durante o processo de cálculo, são elas:

* Divisão: Realiza a divisão de vetores, onde o primeiro argumento é o dividendo e o segundo o divisor, abaixo um exemplo de uso:
  + rca\_numerator = np.divide(bra\_tbl, col\_sums)

Como uma divisão pode gerar infinito ou ser necessário ter algo diferente de 0, o Numpy possibilita criar campos com os valores:

* Valor Vazio: Seta nula em valores que precisam ser tratados como tal.
  + Neste exemplo tudo que for zero trocamos por nulo
  + rcas\_dom = rcas\_dom.replace(0, np.nan)
* Valor Inifito: Possível também ter um valor que representa o infinito
  + Neste exemplo verificamos se um determinado valor é infinito
  + mcp[mcp == np.inf] = 0

Funcionalidades específicas para Log podem ser usadas também nesta biblioteca, trabalhando com entrada e saída em array. Abaixo temos um exemplo de execução desta função com um vetor contendo valor 0, 1 e o número Neperiano.

* np.log([1, np.e, np.e\*\*2, 0])
* array([ 0., 1., 2., -Inf])
  + Veja que o número Neperiano pode ser utilizado através do np.e

No Dataviva esta funcionalidade é utilizada para criar uma escala logarítmica de valores de complexidade, utilizado no gráfico de Scatter.

* entropy = tbl.pivot(index=index, columns=column, values="num\_emp").fillna(0)
* entropy = entropy.T / entropy.T.sum()
* entropy = entropy \* np.log(entropy)
* entropy = entropy.sum() \* -1
* es = pd.Series([np.e]\*len(entropy), index=entropy.index)

* + 1. Pandas

Existem funções específicas da biblioteca Pandas, utilizadas durante o processo de cálculo, que serão detalhadas neste capítulo. Os módulos utilizados são:

* import pandas as pd
* import pandas.io.sql as sql

*Read\_csv*: Por meio desta função realiza-se a leitura de arquivos textos, em formato CSV, resultando em uma matriz de dados.

* ybi\_file\_path = os.path.abspath(os.path.join(DATA\_DIR, 'rais', year, 'ybi.tsv'))
* ybi\_file\_path = get\_file(ybi\_file\_path)
* ybi = pd.read\_csv(ybi\_file\_path, sep="\t")

*DataFrame*: # merge print "merging datasets..."

* ybio\_required = pd.DataFrame(ybio\_required, columns=["year", "bra\_id", "isic\_id", "cbo\_id", "required"])

*Merge*: Realização de unificação de dois DataFrames, utilizando os seguintes parâmetros.

* O primeiro e segundo parâmetros são os dois Dataframes a serem unificados.
* O terceiro parâmetro é o on, ou seja, quais campos serão utilizados para unificar, identificando duplicatas
* Por último especifica como será feita essa união, seja por meio de outer ou inner join.

Exemplo de uma utilização de merge feito pelo Dataviva:

* ybio = pd.merge(ybio, ybio\_required, on=["year", "bra\_id", "isic\_id", "cbo\_id"], how="outer").fillna(0)

Agora um exemplo de dois pequenos datasets, a sua execução e o resultado encontrado.

* left
  + key lval
  + 0 foo 1
  + 1 foo 2
* right
  + key rval
  + 0 foo 4
  + 1 foo 5
* merge(left, right, on='key')
  + key lval rval
  + 0 foo 1 4
  + 1 foo 1 5
  + 2 foo 2 4
  + 3 foo 2 5

MultiIndex:

* ybi = ybi.reset\_index(level="year")
* ybi\_index = [i for i in ybi.index if len(i[0]) == geo\_level and len(i[1]) == 5]
* ybi\_index = pd.MultiIndex.from\_tuples(ybi\_index, names=["bra\_id", "isic\_id"])
* ybi = ybi.reindex(index=ybi\_index)

Concat:

* ecis = pd.concat(ecis)

Series:

* es = pd.Series([np.e]\*len(entropy), index=entropy.index)

*Read Frame*:

* q = "select year, hs\_id, rca from comtrade\_ypw where year = {0} and "\
* "wld\_id = 'sabra'".format(year)
* bra\_rcas = sql.read\_frame(q, db, index\_col=["year", "hs\_id"])

*Not Null*:

* yp = yp[pd.notnull(yp.pci)]

1. RAIS

Dentro da pasta scripts/rais temos arquivos de script em *Python*, com o código do processo de cálculo e também arquivos Bash, com a automação da chamada destes scripts.

Abaixo temos uma tabela com o resumo de todo o processo de cálculo e a seguir explicaremos passo a passo. Os passos 1 e 2, Agregação e Desagregação, são bem similares e comuns aos processos da RAIS e SECEX, por isso foram comentados anteriormente e não estão detalhados neste capítulo, começaremos então explicando o passo 3.

|  |  |  |  |
| --- | --- | --- | --- |
| Passo | Entrada | Observações | Saída |
| 1 | Raw File | Cria a tabela com todas as agregações juntas, a YBIO   * State, Meso, Planning Region, Munic * 1-Digit, 3-Digit, and 5-Digit ISICs * 1-Digit, 2-Digit, and 4-Digit CBOs | ybio.tsv |
| 2 | ybio.tsv | Cria tabelas com as diferentes agregações possíveis | ybo.tsv  ybi.tsv  yio.tsv  yo.tsv  yi.tsv  yb.tsv |
| 3 | ybio.tsv  ybi.tsv | Calcula o número de empregados recomendado | ybio\_required.tsv |
| 4 | yio.tsv | Calcula a importância | yio\_importance.tsv |
| 5 | ybi.tsv  ybo.tsv  yio.tsv | Calcula as ocupações e indústrias únicas | yi\_uniques.tsv  yo\_uniques.tsv  yb\_uniques.tsv |
| 6 | ybi.tsv | Calcula RCA, Distância e Ganho de Oportunidade | ybi\_rcas\_dist\_opp.tsv |
| 7 | ybio\_required.tsv  ybo.tsv  ybi\_rcas\_dist\_opp.tsv  yio\_importance.tsv  yo\_uniques.tsv  yi\_uniques.tsv  yb\_uniques.tsv | Calcula a taxa de crescimento para salário e número de empregados (8 no total) | ybio\_required\_growth.tsv  ybo\_growth.tsv  ybi\_rcas\_dist\_opp\_growth.tsv  yio\_importance\_growth.tsv  yo\_uniques\_growth.tsv  yi\_uniques\_growth.tsv  yb\_uniques\_growth.tsv |
| 8 | ybio\_required\_growth.tsv  ybo\_growth.tsv  ybi\_rcas\_dist\_opp\_growth.tsv  yio\_importance\_growth.tsv  yo\_uniques\_growth.tsv  yi\_uniques\_growth.tsv  yb\_uniques\_growth.tsv | Por fim, importa os dados calculados para o MYSQL | Banco de dados MYSQL |

* 1. Indicador Recomendado

No passo três já temos o início do cálculo em sí, já que até o momento tivemos apenas preparação da base crua em base agregada e desagregada. Como saída temos um arquivo TSV que irá ser utilizado como insumo nas próximas etapas. Para funcionar é necessário informar um ano e assim o script irá passar por cada localização geográfica calculando o número recomendado de empregados para o YBIO.

* 1. Importância

Calcula qual a importância de uma determinada ocupação para uma indústria, dado um determinado ano. Iniciamos abrindo os arquivos:

* YBIO\_required.tsv
* Get\_all\_cbo

*Get\_ybi\_rcas*: Realiza o cálculo do RCA utilizando a massa salarial. Retorna a matriz com valores de RCA para a linha Localidade, coluna a Atividade e valor a massa salarial.

|  |  |  |
| --- | --- | --- |
|  | Mineração | Agricultura |
| BH | 1 | 0 |
| BETIM | 0 | 1 |

*Denominador*: Com a matriz de RCA para as Localidade x Atividade , é realizado uma somatória de RCA considerando o eixo Atividade. Ou seja, cria-se um vetor com o total de RCA para uma determinada Atividade Econômica.

Exemplo:

|  |  |
| --- | --- |
| Mineração | Agricultura |
| 4 | 2 |

*Painel*: Utilizando a tabela de YBIO, retiram-se colunas desnecessárias e realiza um pivot para termos enfim uma matriz com Localidade na coluna, na linha Atividade e Ocupação, e nos valores o número de empregados.

|  |  |  |
| --- | --- | --- |
|  | BH | BETIM |
| [Agricultura, Adminstrador] | 100 | 200 |
| [Agricultura, Contador] | 233 | 0 |
| [Mineração, Adminstrador] | 34 | 56 |
| [Mineração, Contador] | 56 | 98 |

Para cada Ocupação encontrada: (vamos utilizar como exemplo Administrador)

* *Numerador*: Utilizando o Painel, pegamos os valores apenas desta ocupação e multiplicamos pela matriz de RCA (no exemplo acima: *Get\_ybi\_rcas* ) . Temos como resultado final uma nova matriz que passa a ser truncada, onde o valor final for maior ou igual que 1, vira 1, caso contrário 0.

Empregados:

|  |  |  |
| --- | --- | --- |
|  | BH | BETIM |
| Agricultura | 100 | 200 |
| Mineração | 34 | 0 |

RCA:

|  |  |  |
| --- | --- | --- |
|  | Mineração | Agricultura |
| BH | 1 | 0 |
| BETIM | 0 | 1 |

Resultado:

|  |  |  |
| --- | --- | --- |
|  | Mineração | Agricultura |
| BH | 1 | 0 |
| BETIM | 0 | 0 |

Com o resultado para as Localidade x Atividade, é realizado uma somatória, considerando o eixo Atividade. Ou seja, cria-se um vetor com o total encontrado para uma determinada Atividade Econômica.

|  |  |
| --- | --- |
| Mineração | Agricultura |
| 1 | 0 |

A importância é encontrada dividindo o vetor de numerador pelo denominador, conforme abaixo.

|  |  |
| --- | --- |
| Mineração | Agricultura |
| 1/4 | 0/2 = 0 |

* 1. RCA, Distribuição e Oportunidade

Temos aqui o cálculo dos indicadores básicos do Atlas, teoria do Product Space. Desta forma utiliza-se o módulo criado em python para esta aplicação, chamada Growth. Neste momento temos

* rcas = growth.rca(ybi)
* dist = growth.distance(rcas\_binary, prox).fillna(0)
* prox = growth.proximity(rcas\_binary)
* pci = growth.complexity(rcas\_binary)[1]
* opp\_gain = growth.opportunity\_gain(rcas\_binary, prox, pci)

1. SECEX

Dentro da pasta scripts/secex temos arquivos de script em *Python*, com o código do processo de cálculo e também arquivos Bash, com a automação da chamada destes scripts.

Abaixo temos uma tabela com o resumo de todo o processo de cálculo e a seguir explicaremos passo a passo. Os passos 1 e 2, Agregação e Desagregação, são bem similares e comuns aos processos da RAIS e SECEX, por isso foram comentados anteriormente e não estão detalhados neste capítulo, começaremos então explicando o passo 3.

|  |  |  |  |
| --- | --- | --- | --- |
| Step | Input | Purpose | Output |
| 1 | Raw File | Aggregated YBPW file   * State, Meso, Planning Region, Munic * 2-Digit, 4-Digit, and 6-Digit HS * 2-Digit, and 5-Digit WLDs | ybpw.tsv |
| 2 | ybpw.tsv | Create aggregate table files | ybw.tsv  ybp.tsv  ypw.tsv  yw.tsv  yp.tsv  yb.tsv |
| 3 | yp.tsv  yw.tsv  comtrade\_ecis.tsv  comtrade\_pcis.tsv | Calculate PCIs and WLD ECIs | yp\_pcis.tsv  yw\_ecis.tsv |
| 4 | yp\_pcis.tsv  yb.tsv  ybp.tsv | Calculate BRA ECIs | yb\_ecis.tsv |
| 5 | yp\_pcis.tsv  comtrade\_ypw.tsv | Calculate International RCAs for Brazil | yp\_pcis\_rca.tsv |
| 6 | ybp.tsv  yp\_pcis\_rca.tsv | Calculate Domestic and International RCA, Distance, and Opportunity Gain (6 total) | ybp\_rcas\_dist\_opp.tsv |
| 7 | ybpw.tsv  ybw.tsv  ybp\_rcas\_dist\_opp.tsv  ypw.tsv  yw\_ecis.tsv  yp\_pcis\_rca.tsv  yb\_ecis.tsv | Calculate Trade Value Growth (4 total) | ybpw\_growth.tsv  ybw\_growth.tsv  ybp\_rcas\_dist\_opp\_growth.tsv  ypw\_growth.tsv  yw\_ecis\_growth.tsv  yp\_pcis\_rca\_growth.tsv  yb\_ecis\_growth.tsv |
| 8 | ybpw\_growth.tsv  ybw\_growth.tsv  ybp\_rcas\_dist\_opp\_growth.tsv  ypw\_growth.tsv  yw\_ecis\_growth.tsv  yp\_pcis\_rca\_growth.tsv  yb\_ecis\_growth.tsv | Import Data into MYSQL database |  |
|  |  |  |  |

* 1. Indicadores de Complexidade: Observatory

Neste caso não são realizados cálculos, mas apenas leitura de indicadores de complexidade vindos do *Observatory*. Os indicadores incluídos no *Dataviva* são:

* PCI: Product Complexity Index
* WLD: World Complexity Index
  1. Indicadores de Complexidade Econômica

O cálculo do Indicador de Complexidade Economica (ECI – em inglês: Ecnonomic Complexity Index) é realizado para cada localidade, incluindo os níveis geográficos. Portanto o script percorre todos estes níveis e calcula o RCA para cada um. Daí faz-se a leitura do YBP pegando o do código produto (hs\_id) mais desagregado e então para o YB do nível geográfico da vez.

* hs\_criterion = ybp['hs\_id'].map(lambda x: len(x) == 6)
* bra\_criterion = ybp['bra\_id'].map(lambda x: len(x) == geo\_level)

Assim podemos criar uma tabela pivot com a localidade na linha, produto na coluna (hs\_id) e como valor o total exportado.

|  |  |  |
| --- | --- | --- |
|  | Minério | Arroz |
| BH | 1223,00 | 4566,00 |
| BETIM | 234,00 | 23466,00 |

Com esta entrada utiliza-se o módulo Growth para calcular o RCA, aplicando o truncamento de valor maior que 0 para 1 e em caso contrário 0.

* rcas = growth.rca(ybp)

|  |  |  |
| --- | --- | --- |
|  | Minério | Arroz |
| BH | 1 | 1 |
| BETIM | 1 | 0 |

Outra informação necessária para este cálculo é o PCI, ou seja, o índice de complexidade para o produto dado uma determinada localidade e ano.

|  |  |  |
| --- | --- | --- |
|  | Minério | Arroz |
| PCI | 1,22 | 0,44 |

Com estas duas informações, fazemos a multiplicação do RCA para com o índice de complexidade encontrado, resultando na listagem de PCI das cidades com RCA>1.

|  |  |  |
| --- | --- | --- |
|  | Minério | Arroz |
| BH | 1,22 | 0,44 |
| BETIM | 1,22 | 0 |

Fazemos a soma de todos os PCI’s e RCA para determinada localidade:

|  |  |
| --- | --- |
| BH | BETIM |
| 1,66 | 1,22 |

|  |  |  |
| --- | --- | --- |
|  | BH | BETIM |
| RCA | 2 | 1 |

Por fim fazemos a divisão destas duas, obtendo:

|  |  |  |
| --- | --- | --- |
|  | BH | BETIM |
| ECI | 1,66 / 2 = 0,83 | 1,22 /1 = 1,22 |

* 1. RCA, Distribuição e Oportunidade

Temos aqui o cálculo dos indicadores básicos do *Atlas*, teoria do *Product Space*. Desta forma utiliza-se o módulo criado em *Python* para esta aplicação, chamada *Growth*.

Para este cálculo é necessário as bases de *RCA* para domínio doméstico e internacional, truncados para se houver RCA>=1 ganha valor 1 e caso contrário 0.

*RCA Internacional*: O valor internacional vem da tabela comtrade\_ypw, portanto sem realização de cálculos.

*RCA Doméstico*: Já este é obtido através da tabela de dados ybp.tsv e repassado ao growth lib. Para isso é feito o pivot desta ybp.tsv considerando a linha como localidade, coluna como produto (hd\_is) e o total de exportação como valor.

* ybp = ybp.pivot(index="bra\_id", columns="hs\_id", values="val\_usd").fillna(0)

|  |  |  |
| --- | --- | --- |
|  | Minério | Arroz |
| BH | 15675,22 | 234,44 |
| BETIM | 1234,22 | 0 |

O resultado desta tabela, após ser processada pelo Growth Lib é:

* rcas = growth.rca(ybp)

|  |  |  |
| --- | --- | --- |
|  | Minério | Arroz |
| BH | 1 | 1 |
| BETIM | 1 | 0 |

Para o cálculo da distância, obtém-se antes a proximidade por meio da Growth Lib e a tabela mostrada acima.

* prox\_dom = growth.proximity(rcas\_dom\_binary)

|  |  |  |
| --- | --- | --- |
|  | Minério | Arroz |
| Minério | 1 | 0,56 |
| Arroz | 0,22 | 1 |

Com a tabela de proximidade e RCA, calculamos a distância, lembrando que este cálculo é realizado para doméstico e internacional:

* dist\_dom = growth.distance(rcas\_dom\_binary, prox\_dom).fillna(0)

O ganho de oportunidade utiliza a mesma lógica, através do Growth Lib e para os dois domínios, doméstico e internacional:

* opp\_gain\_dom = growth.opportunity\_gain(rcas\_dom\_binary, prox\_dom, pcis\_dom)

1. HOMOLOGAÇÃO

Etapas do processo de Homologação:

RAIS:

|  |  |  |
| --- | --- | --- |
| Etapa | Descrição | Responsável |
| 1 | Checar número de empregados e massa salarial por município com dados do MTE | NAAI |
| 2 | Checar número de empregados e massa salarial por ocupação com dados do MTE | NAAI |
| 3 | Checar número de empregados e massa salarial por atividade econômica com dados do enviados para a GV | Núcleo de Sistema e Gestão |

SECEX

|  |  |  |
| --- | --- | --- |
| Etapa | Descrição | Responsável |
| 1 | Checar valor das exportações por município e estado com base nos dados enviados a GV | NAAI |
| 2 | Checar valor das exportações por produto com base nos dados enviados a GV | NAAI |
| 3 | Checar valor das exportações por destino com base nos dados enviados a GV | NAAI |

GERAL:

|  |  |  |
| --- | --- | --- |
| Etapa | Descrição | Responsável |
| 1 | Checar número de empregados e massa salarial por município com dados do MTE | Núcleo de Sistema e Gestão |

* 1. Agregações

Temos diversas tabelas sobre o mesmo assunto, seja para SECEX ou RAIS. Isto acontece pois é realizado as combinações das dimensões existentes, deixando tabela de dados prontas para serem filtradas. Desta forma é retirado o processamento de agregação da aplicação e passa a ser feito previamente, para o preenchimento das tabelas.

Sendo assim, se tomarmos como exemplo a RAIS, temos as dimensões YBIO (Year – Local – Industry and Occupation)

* 1. Localidades

Aqui fazemos comparações entre as tabelas agregadas e desagregadas, considerando o eixo localidade. Portanto a ideia é utilizar o script abaixo para comparar a YB com a tabela YBIO, YBI, YBO.

YBIO:

select \*

from (

select

YBIO.num\_emp - YB.num\_emp as num\_empDiff,

YBIO.wage - YB.wage as wageDiff

, YB.bra\_id, YB.year

from (

select

sum(wage) as wage,sum(num\_emp) as num\_emp

,bra\_id,year

from rais\_yb

group by bra\_id,year

) as YB,

(

select

sum(wage) as wage,sum(num\_emp) as num\_emp

,bra\_id,year

from rais\_ybio where CHAR\_LENGTH(cbo\_id) =4

group by bra\_id,year

) as YBIO

where YBIO.year = YB.year and YBIO.bra\_id = YB.bra\_id

) as DiffYBIOxYB

where num\_empDiff<>0 or wageDiff<>0

YBO:

select \*

from (

select

YBO.num\_emp - YB.num\_emp as num\_empDiff,

YBO.wage - YB.wage as wageDiff

, YB.bra\_id, YB.year

from (

select

sum(wage) as wage,sum(num\_emp) as num\_emp

,bra\_id,year

from rais\_yb

group by bra\_id,year

) as YB,

(

select

sum(wage) as wage,sum(num\_emp) as num\_emp

,bra\_id,year

from rais\_ybo where CHAR\_LENGTH(cbo\_id) =4

group by bra\_id,year

) as YBIO

where YBO.year = YB.year and YBO.bra\_id = YB.bra\_id

) as DiffYBOxYB

where num\_empDiff<>0 or wageDiff<>0

YBI:

select \*

from (

select

YBI.num\_emp - YB.num\_emp as num\_empDiff,

YBI.wage - YB.wage as wageDiff

, YB.bra\_id, YB.year

from (

select

sum(wage) as wage,sum(num\_emp) as num\_emp

,bra\_id,year

from rais\_yb

group by bra\_id,year

) as YB,

(

select

sum(wage) as wage,sum(num\_emp) as num\_emp

,bra\_id,year

from rais\_ybi where CHAR\_LENGTH(isic\_id) = 5

group by bra\_id,year

) as YBI

where YBI.year = YB.year and YBI.bra\_id = YB.bra\_id

) as DiffYBIxYB

where num\_empDiff<>0 or wageDiff<>0

* 1. Indústrias

Aqui fazemos comparações entre as tabelas agregadas e desagregadas, considerando o eixo indústria. Portanto a ideia é utilizar o script abaixo para comparar a YI com a tabela YBIO, YBI,YIO.

YBIO:

select \*

from (

select

YBIO.num\_emp - YI.num\_emp as num\_empDiff,

YBIO.wage - YI.wage as wageDiff

, YI.isic\_id, YI.year

from (

select

sum(wage) as wage,sum(num\_emp) as num\_emp

, isic\_id,year

from rais\_yi where CHAR\_LENGTH(isic\_id) =5

group by isic\_id,year

) as YI,

(

select

sum(wage) as wage,sum(num\_emp) as num\_emp

,isic\_id,year

from rais\_ybio where CHAR\_LENGTH(isic\_id) =5

group by isic\_id,year

) as YBIO

where YBIO.year = YI.year and YBIO.isic\_id = YI.isic\_id

) as DiffYBIOxYI

where num\_empDiff<>0 or wageDiff<>0

YBI:

select \*

from (

select

YBI.num\_emp - YI.num\_emp as num\_empDiff,

YBI.wage - YI.wage as wageDiff

, YI.isic\_id, YI.year

from (

select

sum(wage) as wage,sum(num\_emp) as num\_emp

, isic\_id,year

from rais\_yi where CHAR\_LENGTH(isic\_id) =5

group by isic\_id,year

) as YI,

(

select

sum(wage) as wage,sum(num\_emp) as num\_emp

,isic\_id,year

from rais\_ybi where CHAR\_LENGTH(isic\_id) =5

group by isic\_id,year

) as YBI

where YBIO.year = YI.year and YBIO.isic\_id = YI.isic\_id

) as DiffYBIOxYI

where num\_empDiff<>0 or wageDiff<>0

* 1. Ocupações

Aqui fazemos comparações entre as tabelas agregadas e desagregadas, considerando o eixo indústria. Portanto a ideia é utilizar o script abaixo para comparar a YI com a tabela YBIO, YBI,YIO.

* 1. Banco de Dados

A base final do Sistema é utilizando o banco de dados MYSQL e portanto precisamos extrair informações dele para realizar uma homologação completa. Para isso existem algumas técnicas do MYSQL necessárias para este processo.

O principal é como executar comandos SQL que podem demorar muito tempo para executar, muito comum em cortes realizado na RAIS. Veja abaixo um exemplo de conexão com o banco e extreção de um resultado de SQL em um arquivo texto:

mysql -uroot -pmysql ps\_launch -e"select s.year, a.id\_ibge, a.name\_pt, a.id, s.hs\_id, s.wld\_id,w.id\_mdic, w.id\_num, sum(val\_usd) as fob from secex\_ybpw s, attrs\_bra a,attrs\_wld w where s.bra\_id = a.id and w.id = s.wld\_id group by s.year, a.id\_ibge, a.name\_pt, a.id, s.hs\_id, s.wld\_id, w.id\_mdic, w.id\_num" > test.txt

* 1. RAIS

O formato de extração do banco da RAIS, a ser enviado a equipe do NAAI para continuidade da homologação, deve seguir o padrão:

* Ano
* código IBGE do município, Código da mesorregião ou   “\N”  para UF
* Nome da região ou Nome da UF
* Identificador utilizado unicamente pelo DATAVIVA para identificar as regionalizações
* Ocupação CBO
* Frequência
* Massa salarial

O script SQL para realizar esta extração é:

select

r.year,

a.id\_ibge,

a.name\_pt,

a.id,

r.cbo\_id,

sum(num\_emp) as num\_emp,

sum(wage) as wage

from rais\_ybo r, attrs\_bra a where r.bra\_id = a.id

group by r.year,

a.id\_ibge,

a.name\_pt,

a.id,

r.cbo\_id;

Abaixo o script pronto para rodar via command:

mysql -uroot -pmysql ps\_launch -e"select r.year,a.id\_ibge,a.name\_pt,a.id,r.cbo\_id, sum(num\_emp) as num\_emp, sum(wage) as wage from rais\_ybo r, attrs\_bra a where r.bra\_id = a.id group by r.year,a.id\_ibge,a.name\_pt,a.id,r.cbo\_id" > testRais.txt

* + 1. ISIC

Durante o processo de extração e transformação da base de dados recebida pelo MTE, são criados relatórios com valores agrupados por ISIC. Desta forma temos relatórios com valores agrupados considerando a base original, ideal para ser comparada com um agrupamento retirado do banco do Dataviva. Os campos necessários para esta comparação é:

* Ano,
* ID\_IBGE (Municipality\_ID)
* ISIC (EconomicAtivity\_ID\_ISIC)
* Empregados
* Salário (AverageMonthlyWage)

A consulta para obter este valor do banco MySQL é:

select

r.year,a.id\_ibge,r.isic\_id,

sum(num\_emp) as num\_emp, sum(wage) as wage

from rais\_ybi r, attrs\_bra a

where r.bra\_id = a.id

group by r.year,a.id\_ibge,r.isic\_id

Aqui o script para execução via command:

mysql -uroot -pmysql ps\_launch -e"select r.year,a.id\_ibge,r.isic\_id, sum(num\_emp) as num\_emp, sum(wage) as wage from rais\_ybio r, attrs\_bra a where r.bra\_id = a.id group by r.year,a.id\_ibge,r.isic\_id" > testRaisISIC.txt

Por fim, podemos fazer o comparativo de um com outro, utilizando o script do QLikview , através da operação de homologação.

* 1. SECEX

Parte da homologação da SECEX é realizada em um software estatístico. Para isso precisamos fazer uma extração da base considerando as seguintes informações:

* Ano,
* código IBGE do município, Código da mesorregião ou   código UF definido pelo MDIC
* Nome da região ou da UF
* Identificador utilizado unicamente pelo DATAVIVA para identificar as regionalizações
* Código HS customizado pelo DATAVIVA
* FOB
* País - código

Para realizar o corte destas informações do banco DataViva, deve-se utilizar o script abaixo:

select

s.year, a.id\_ibge, a.name\_pt, a.id, s.hs\_id, s.wld\_id,w.id\_mdic, w.id\_num, sum(val\_usd) as fob

from secex\_ybpw s, attrs\_bra a,attrs\_wld w

where s.bra\_id = a.id and w.id = s.wld\_id

group by s.year, a.id\_ibge, a.name\_pt, a.id, s.hs\_id, s.wld\_id, w.id\_mdic, w.id\_num

Para sabermos valores agrupados do MDIC, pegando da base original que o Escritório recebeu, podemos executar os scripts abaixo:

load

ANO,

NCM,

sum( num(replace(ltrim(replace(VALORFOB,0,' ')),' ',0))) ,

MUNICÍPIO as MunIDMDIC

from

[F:\Banco de dados\MDIC\Dados originais\EPEMG\_BREX20.qvd] (qvd)

group by ANO, NCM , MUNICÍPIO;

load

ANO,

NCM, // PAIS,

MUNICÍPIO as MunIDMDIC,

sum(Num#(trim(VALORFOB),'#')) as valorSecex

,MaxString(VALORFOB),

replace(ltrim(replace(VALORFOB,0,' ')),' ',0) as test //sum(num(trim(VALORFOB))) as valorSecex

from [F:\Banco de dados\MDIC\Dados originais\EPEMG\_BREX20.qvd] (qvd)

group by ANO, NCM , MUNICÍPIO;