**DataViva**

Processo de Cálculo

V1.2

Núcleo de Sistemas e Gestão

Novembro 2014

Histórico de Revisão

|  |  |  |  |
| --- | --- | --- | --- |
| DATA | VERSÃO | DESCRIÇÃO | AUTOR |
| 29/07/2011 | 1.0 | Criação do documento | Mario H. Mol |
| 17/10/2014 | 1.1 | Início da Atualização Dataviva 2 | Mario H. Mol |
| 29/07/2011 | 1.2 | Atualizado Dataviva 2 | Mario H. Mol |

ÍNDICE

[1. INTRODUÇÃO 5](#_Toc404186715)

[2. ARQUITETURA 6](#_Toc404186716)

[2.1. Dependências 6](#_Toc404186717)

[2.2. Growth Lib 7](#_Toc404186718)

[2.3. Configuração 8](#_Toc404186719)

[2.4. Execução 8](#_Toc404186720)

[3. PROCESSOS 10](#_Toc404186721)

[3.1. Agregação 10](#_Toc404186722)

[3.2. Desagregação 11](#_Toc404186723)

[3.3. Taxa e Valor de Crescimento 12](#_Toc404186724)

[3.1. Diversidade 13](#_Toc404186725)

[3.2. Funções 14](#_Toc404186726)

[3.2.1. NumPy 15](#_Toc404186727)

[3.2.2. Pandas 16](#_Toc404186728)

[3.3. Arquitetura 18](#_Toc404186729)

[4. RAIS 19](#_Toc404186730)

[4.1. Agregação 19](#_Toc404186731)

[4.2. Importância 20](#_Toc404186732)

[4.3. Diversidade 21](#_Toc404186733)

[4.4. RCA e Oportunidade 22](#_Toc404186734)

[4.5. Geral 22](#_Toc404186735)

[4.6. Cálculo 23](#_Toc404186736)

[4.7. Importar 23](#_Toc404186737)

[5. SECEX 24](#_Toc404186738)

[5.1. Agregação 24](#_Toc404186739)

[5.2. Indicadores de Complexidade: Observatory 25](#_Toc404186740)

[5.3. Indicadores de Complexidade Econômica 25](#_Toc404186741)

[5.4. RCA, Distribuição e Oportunidade 26](#_Toc404186742)

[6. EI 29](#_Toc404186743)

[6.1. Geral 29](#_Toc404186744)

[6.2. Importar 29](#_Toc404186745)

[7. Higher EducatioN 31](#_Toc404186746)

[7.1. Geral 31](#_Toc404186747)

[7.2. Cálculos 32](#_Toc404186748)

[7.3. Importar 32](#_Toc404186749)

[8. School census 33](#_Toc404186750)

[8.1. Geral 33](#_Toc404186751)

[8.2. Cálculos 34](#_Toc404186752)

[8.3. Importar 34](#_Toc404186753)

[9. HOMOLOGAÇÃO 35](#_Toc404186754)

[9.1. Agregações 35](#_Toc404186755)

[9.2. Localidades 36](#_Toc404186756)

[9.3. Indústrias 37](#_Toc404186757)

[9.4. Ocupações 38](#_Toc404186758)

[9.5. Banco de Dados 39](#_Toc404186759)

[9.6. RAIS 39](#_Toc404186760)

[9.6.1. ISIC 40](#_Toc404186761)

[9.7. SECEX 41](#_Toc404186762)

[9.8. Higher Education 42](#_Toc404186763)

[9.9. School Census 42](#_Toc404186764)

[9.10. Eletronic Invoice 42](#_Toc404186765)

INTRODUÇÃO

Foi implantado em Minas Gerais o *Dataviva* (DV), em parceria com a *Datawheel* (DW) 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 DW entregaram ao Escritório de Prioridades Estratégicas de Minas Gerais (EPE-MG) o fonte deste sistema e conhecimento para que seja possível dar continuidade na sua manutenção e evolução. Utilizando este fonte foram realizadas análises 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/DataViva/dataviva-scripts/>
  + https://github.com/DataViva/dataviva-scripts/tree/master/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
  + **common**: Funções comuns a todos os passos de processamento ficam nesta pasta
  + **comtrade**: Operações de cálculo relacionado a base de dados COMTRADE
  + **ei**: Eletronic Invoice, portanto scripts da Nota Fiscal Eletrônica
  + **growth\_lib**: Biblioteca que realiza cálculo de indicadores do Product Space, que aponta para outro projeto:
    - https://github.com/alexandersimoes/ps\_calcs
  + **hedu**: Processo de cálculo para a High Education, portanto dados da educação superior
  + **misc**: Outros métodos e funções a ser utilizado pelos outros módulos de cálculo
  + **rais\_new** : Arquivos com os passos de 1 a 7 referente a execução da Rais
    - O diretório **\script\rais** é relativo ao processo de cálculo do Dataviva1
  + **sc**: School Census, portanto arquivos de cálculo para as bases de educação básica
  + **secex**: Arquivos com passos de 1 a 8 para os cálculos da Secex.
    - O diretório **\script\secex** é relativo ao processo de cálculo do Dataviva1
  + **tests**: Alguns arquivos para verificação dos dados
  1. Dependências

Existem bibliotecas específicas do *Python* que precisam ser instaladas para executar este processo. Vamos começar por quem 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/ps\_calcs

Para configurar no projeto, copie a pasta *growth* que está dentro deste repositório do *github* e cole na pasta do *dataviva-scripts/scripts/growth-lib/.* Existem bibliotecas específicas do *Python* que precisam ser instaladas para executar este processo:

* numpy , matplotlib
* scipy
  + http://www.lfd.uci.edu/~gohlke/pythonlibs/#scipy
* scikits.statsmodels
  + https://pypi.python.org/pypi/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. Devem ser declaradas em um arquivo chamado *config.py* na pasta */scripts/,* as variáveis são:

* **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*. Como temos outras bases de dados e outras formas de agregação, teremos mais tabelas com combinações como estas.

* 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 utilizada 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

Cada processo de cálculo tem suas similaridades e particularidades, seja em qualquer uma das bases envolvidas, de *SECEX* à *Censo Educacional*. Neste capítulo vamos exemplificar as principais etapas de cálculo utilizado nas diversas bases de dados, contendo documentação da 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. Arquitetura

Consideramos como um processo de arquitetura onde este está relacionado a questões de arquitetura de sistemas, portanto puramente técnico. Pode haver uma necessidade de performance ou de organização de dados.

* **Tamanho das colunas**: *Dataviva 2* tem dado agregado ou desagregado, podemos tomar como exemplo a variável de localidade. Temos informações para o município, para micro região, meso região, estados e país. Sendo assim o sistema precisa perguntar ao banco de dados se tal informação é de um nível de agregação, para isso ele consulta o tamanho da coluna de localidade. Se o identificador desta localidade tem apenas 2 caracteres, por exemplo MG, quer dizer que é um estado. E assim por diante, portanto ter no banco de dados o tamanho do campo é muito importante para termos uma alta performance na consulta de dados. Para isso existem processos de cálculo onde realiza a criação desta coluna de tamanho.
* **Identificador de Atributos**: Neste projeto temos tabelas próprias para designar atributos, como por exemplo a tabela *attrs\_bra*, onde temos todos estados, municípios e regiões do Brasil. Estas tabelas criam um identificador único para cada localidade, contendo alguns padrões como por exemplo seu tamanho, quanto menor o número de caracteres. Portanto existe a necessidade de fazer a tradução entre o identificador original da base e o utilizado pelo *Dataviva*. Tomando como exemplo o *attrs\_bra*, fazemos a migração do código de cidade em formato do IBGE para o formato *Dataviva*.

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. Esta pasta é relativo aos scripts de cálculo do *Dataviva* 1, contendo também uma pasta chamada *scripts/rais\_new* com o novo processo de cálculo para a *Rais*.

Abaixo temos uma tabela com o resumo de todo o processo de cálculo e a seguir explicaremos passo a passo.

|  |  |
| --- | --- |
| Passo | Observações |
| 1 | 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 |
| 2 | Cria tabelas com as diferentes agregações possíveis |
| 3 | Calcula o número de empregados recomendado |
| 4 | Calcula a importância |
| 5 | Calcula as ocupações e indústrias únicas |
| 6 | Calcula RCA, Distância e Ganho de Oportunidade |
| 7 | Calcula a taxa de crescimento para salário e número de empregados (8 no total) |
| 8 | Por fim, importa os dados calculados para o MYSQL |

Existem alguns scripts que estão em análise de viabilidade e eles hoje não são utilizados, portanto ainda em dúvida. O caso do required está sendo discutido qual metodologia utilizar e portanto ainda não temos uma definição de como será realizado o cálculo:

* <https://github.com/DataViva/dataviva-scripts/blob/master/scripts/rais_new/_required.py>

Outro script que é listado, mas ate o momento não utilizado é o calculado do Shard, abaixo os scripts relacionados nesta operação.

* <https://github.com/DataViva/dataviva-scripts/blob/master/scripts/rais_new/_shard.py>
* <https://github.com/DataViva/dataviva-scripts/blob/master/scripts/rais_new/_shard_old.py>
  1. Agregação

Os passos 1 e 2, Agregação e Desagregação, são bem similares e comuns aos processos da RAIS e SECEX, realizada através do script:

* <https://github.com/DataViva/dataviva-scripts/blob/master/scripts/rais_new/_aggregate.py>
  1. Importância

Calcula qual a importância de uma determinada ocupação para uma indústria, dado um determinado ano. O arquivo que realiza este cálculo é o:

* <https://github.com/DataViva/dataviva-scripts/blob/master/scripts/rais_new/_importance.py>

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. Diversidade

O cálculo da diversidade é realizado pelo script abaixo, onde são criados dois conceitos, a diversidade e a diversidade efetiva. Esta diversidade é calculada para localidade, atividade econômica e ocupação.

* <https://github.com/DataViva/dataviva-scripts/blob/master/scripts/rais_new/_calc_diversity.py>

Os campos criados no banco com essas taxas são:

* bra\_diversity e bra\_diversity\_eff
* cnae\_diversity e cnae\_diversity\_eff
* cbo\_diversity e cbo\_diversity\_eff

A forma como este cálculo funciona é bem simples, vamos demonstrar utilizando a cnae\_diversity. Neste caso o processo verifica se uma determinada atividade econômica em uma tal localidade tem um ou mais empregados, atribuindo o valor 1, ou se não houver empregados atribui 0.

|  |
| --- |
| diversity[diversity >= 1] = 1 |
| diversity[diversity < 1] = 0 |

Com esta tabela binária realiza a soma de atividades econômicas que empregam na localidade, gerando a diversidade.

* 1. RCA 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)

O script responsável por esta etapa é:

* <https://github.com/DataViva/dataviva-scripts/blob/master/scripts/rais_new/_rdo.py>
  1. Geral

Descrevemos aqui etapas do processo de cálculo que são comuns a grande parte das bases de dados. Primeiro temos o script que realiza a contagem do tamanho de campos de atributos e cria uma nova coluna no banco de dados:

* <https://github.com/DataViva/dataviva-scripts/blob/master/scripts/rais_new/_column_lengths.py>

Outro processo é a realização de cálculos de taxa de crescimento, calculado para crescimento anual e plurianual (5 anos), utilizado para as variáveis Salário e Número de empregado. Estas taxas são gravadas nos campos:

* wage\_growth e wage\_growth\_5
* num\_emp\_growth e num\_emp\_growth\_5

O script contendo os métodos utilizados no cálculo da taxa de crescimento:

* <https://github.com/DataViva/dataviva-scripts/blob/master/scripts/rais_new/_growth.py>

Para realizar a execução deste método, existe um script próprio pra isso:

* <https://github.com/DataViva/dataviva-scripts/blob/master/scripts/rais_new/compute_growth.py>
  1. Cálculo

Descrevemos aqui etapas

* <https://github.com/DataViva/dataviva-scripts/blob/master/scripts/rais_new/compute_mne.py>
* <https://github.com/DataViva/dataviva-scripts/blob/master/scripts/rais_new/est_and_emp_uniqueness.py>
* <https://github.com/DataViva/dataviva-scripts/blob/master/scripts/rais_new/format_demographic_data.py>
* <https://github.com/DataViva/dataviva-scripts/blob/master/scripts/rais_new/format_raw_data.py>
  1. Importar

A etapa final é importar os dados já calculados no banco de dados que irá ser utilizado no Dataviva. Conforme os demais processos de cálculo existem duas diferentes formas de importar os dados, seja por meio de um script python ou através de um arquivo de lotes, o bash do Windows.

* <https://github.com/DataViva/dataviva-scripts/blob/master/scripts/rais_new/helpers/import.py>
* <https://github.com/DataViva/dataviva-scripts/blob/master/scripts/rais_new/helpers/db_importer.sh>

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 |
| 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 |
| 2 | ybpw.tsv | Create aggregate table files |
| 3 | yp.tsv  yw.tsv  comtrade\_ecis.tsv  comtrade\_pcis.tsv | Calculate PCIs and WLD ECIs |
| 4 | yp\_pcis.tsv  yb.tsv  ybp.tsv | Calculate BRA ECIs |
| 5 | yp\_pcis.tsv  comtrade\_ypw.tsv | Calculate International RCAs for Brazil |
| 6 | ybp.tsv  yp\_pcis\_rca.tsv | Calculate Domestic and International RCA, Distance, and Opportunity Gain (6 total) |
| 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) |
| 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. Agregação

Os passos 1 e 2, Agregação e Desagregação

* <https://github.com/DataViva/dataviva-scripts/blob/master/scripts/secex_new/_aggregate.py>
  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
* <https://github.com/DataViva/dataviva-scripts/blob/master/scripts/secex_new/_brazil_rca.py>
  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.

O script responsável por esta etapa:

* <https://github.com/DataViva/dataviva-scripts/blob/master/scripts/secex_new/_domestic_eci.py>

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)

* <https://github.com/DataViva/dataviva-scripts/blob/master/scripts/secex_new/_merge.py>
* <https://github.com/DataViva/dataviva-scripts/blob/master/scripts/secex_new/_pci_wld_eci.py>
* <https://github.com/DataViva/dataviva-scripts/blob/master/scripts/secex_new/_rdo.py>
* <https://github.com/DataViva/dataviva-scripts/blob/master/scripts/secex_new/_shard.py>
* <https://github.com/DataViva/dataviva-scripts/blob/master/scripts/secex_new/_to_df.py>
* <https://github.com/DataViva/dataviva-scripts/blob/master/scripts/secex_new/_val_per_unit.py>
* <https://github.com/DataViva/dataviva-scripts/blob/master/scripts/secex_new/dominant_unit.py>
* <https://github.com/DataViva/dataviva-scripts/blob/master/scripts/secex_new/format_raw_data.py>
* <https://github.com/DataViva/dataviva-scripts/blob/master/scripts/secex_new/_calc_diversity.py>
  1. Geral

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

* <https://github.com/DataViva/dataviva-scripts/blob/master/scripts/secex_new/_column_lengths.py>
* <https://github.com/DataViva/dataviva-scripts/blob/master/scripts/secex_new/_growth.py>
  1. Cálculo

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

* 1. Importar

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

1. EI

Dentro da pasta scripts/ei temos arquivos de script em *Python*, com o código do processo de cálculo e também arquivos Bash, para a execução da base de dados de Nota Fiscal Eletrônica, no inglês Eletronic Invoice (EI).

Abaixo temos uma tabela com o resumo de todo o processo

* 1. Geral

*Helper Columns*: Após todo o processamento algumas colunas auxiliares devem ser criadas, pensando principalmente em desempenho. Portanto este arquivo faz a criação destas colunas, como a coluna de tamanho da localidade bra\_id\_len

* https://github.com/DataViva/dataviva-scripts/blob/master/scripts/ei/\_helper\_columns.py

*0 Month*: No dataviva2 foi criado o conceito de mês 0, isso indica que é o valor relativo a todo o ano informado. Portanto este script faz a agregação dos meses por ano e a criação do registro com o mês 0.

* https://github.com/DataViva/dataviva-scripts/blob/master/scripts/ei/finalize\_0\_month.py

*Table Agregator*: ??

* https://github.com/DataViva/dataviva-scripts/blob/master/scripts/ei/table\_aggregator.py

*Format Raw Data*:

* https://github.com/DataViva/dataviva-scripts/blob/master/scripts/ei/format\_raw\_data.py
  1. Importar

A tabela do banco de dados para importação dos dados deve ser criada com o seguinte script:

* https://github.com/DataViva/dataviva-scripts/blob/master/scripts/ei/ei\_create\_tables.sql

O processo de pegar os dados calculados e inserir no banco de dados é chamado de etapa de Import, representado no arquivo import.py

* <https://github.com/DataViva/dataviva-scripts/blob/master/scripts/ei/import.py>
* Forma de chamar, incluindo como argumento a pasta onde estão os arquivos já processados para inclusão no banco de dados do dataviva
  + scripts/ei/import.py --idir data/

Basicamente este arquivo pega todos os arquivos .csv da pasta, realiza uma leitura utilizando separador ‘;’ e então faz inserção no banco de dados linha a linha.

Não existem indicadores com cálculos diferenciados ou qualquer outra aplicação nesta base de dados, restando apenas a consolidação de dados em diferentes agrupamentos e a aplicaç

1. Higher EducatioN

Dentro da pasta scripts/hedu temos arquivos de script em *Python*, com o código do processo de cálculo e também arquivos *Bash*, para realização de cálculo na base de dados de Educação Superior, ou em inglês *Higher Education* (*hedu*).

Existem alguns scripts que estão em fase de desenvolvimento e são ainda considerado testes, portanto temos dúvidas se serão oficializados ou não:

* <https://github.com/DataViva/dataviva-scripts/blob/master/scripts/hedu/generate_cid2.py>
* <https://github.com/DataViva/dataviva-scripts/blob/master/scripts/hedu/experiments.py>
* <https://github.com/DataViva/dataviva-scripts/blob/master/scripts/hedu/_calc_rca.py>
  1. Geral

Existe um script *python* que realiza a inserção dos arquivos gerados durante este processo de cálculo para o banco de dados:

* <https://github.com/DataViva/dataviva-scripts/blob/master/scripts/hedu/_to_df.py>

Aqui temos um caso onde é feito a tradução dos identificadores de localidade e curso de um número utilizado pela base de dados original para um número de identificação próprio do *Dataviva*:

* <https://github.com/DataViva/dataviva-scripts/blob/master/scripts/hedu/_replace_vals.py>

Como desenhado na arquitetura deste projeto, temos algumas variáveis que são muito utilizadas como índice de pesquisa e precisam ter o seu tamanho gravado previamente. Nesta base foi criado o campo com o tamanho para duas variáveis, localidade e curso, e deve ser executado o seguinte arquivo:

* <https://github.com/DataViva/dataviva-scripts/blob/master/scripts/hedu/_column_lengths.py>
  1. Cálculos

A última etapa do processo é realizar o cálculo de agregações por região, ou seja, ter já pré-calculado valores para o micro e mesorregião, além de estado e nível país.

* <https://github.com/DataViva/dataviva-scripts/blob/master/scripts/hedu/_aggregate.py>

A execução do cálculo:

* <https://github.com/DataViva/dataviva-scripts/blob/master/scripts/hedu/format_raw_data.py>
  1. Importar

Existe um script em *Python* que realiza a inserção dos arquivos gerados durante este processo de cálculo para o banco de dados:

* <https://github.com/DataViva/dataviva-scripts/blob/master/scripts/hedu/import.py>

Outro script realiza a criação da estrutura do banco de dados e também pode importar os dados ao banco de dados, mas através de um arquivo de lote em batch, e não um script *Python* como o anterior:

* <https://github.com/DataViva/dataviva-scripts/blob/master/scripts/hedu/import_sql_all.sh>

1. School census

Dentro da pasta *scripts/secex* temos arquivos de script em *Python*, com o código do processo de cálculo e também arquivos *Bash*, para execução da base de dados de Censo Escolar, ou em inglês *School Census* (sc).

Existem alguns scripts que estão em fase de desenvolvimento e são ainda considerados testes, portanto temos dúvidas se serão oficializados ou não:

* <https://github.com/DataViva/dataviva-scripts/blob/master/scripts/sc/_calc_rca.py>
* <https://github.com/DataViva/dataviva-scripts/blob/master/scripts/sc/_shard.py>
  1. Geral

Como desenhado na arquitetura deste projeto, temos algumas variáveis que são muito utilizadas como índice de pesquisa e precisam ter o seu tamanho gravado previamente. Nesta base foi criado o campo com o tamanho para duas variáveis, localidade e curso, e deve ser executado o seguinte arquivo:

* <https://github.com/DataViva/dataviva-scripts/blob/master/scripts/sc/_column_lengths.py>

Aqui temos um caso onde é feito a tradução dos identificadores de localidade de um número utilizado pela base de dados original, no caso aqui o *IBGE*, para um número de identificação próprio do *Dataviva*, chamado de *BRA\_ID*.

* <https://github.com/DataViva/dataviva-scripts/blob/master/scripts/sc/_replace_vals.py>

Além da localidade existe a necessidade de outros mapeamentos, como por exemplo, o identificador do curso ou etnia, para isso temos o script:

* <https://github.com/DataViva/dataviva-scripts/blob/master/scripts/sc/_to_df.py>

As agregações utilizadas nesta base são referentes apenas a localidades (mesorregião, microrregião, estado..), e elas são realizadas no script abaixo. O único ponto de detalhe é que a distorção idade séria é calculado também neste arquivo:

* <https://github.com/DataViva/dataviva-scripts/blob/master/scripts/sc/_aggregate.py>
  1. Cálculos

Para realização de todo o procedimento de cálculo, fazendo chamada para métodos que estão nos demais arquivos citados neste capítulo, temos o arquivo abaixo:

* <https://github.com/DataViva/dataviva-scripts/blob/master/scripts/sc/format_raw_data.py>

Um exemplo de chamada deste script pode ser realizado por meio de:

* *python scripts/edu/format\_raw\_data.py data/edu/School\_census\_2007.csv.bz2 -y 2007 -o data/edu/*
  1. Importar

Existe um script *Python* que realiza a inserção dos arquivos gerados durante este processo de cálculo para o banco de dados:

* <https://github.com/DataViva/dataviva-scripts/blob/master/scripts/sc/import.py>

Outro script realiza a criação da estrutura do banco de dados e também pode importar os dados ao banco de dados, mas através de um arquivo de lote em batch, e não um script *Python* como o anterior:

* <https://github.com/DataViva/dataviva-scripts/blob/master/scripts/sc/import_sql_all.sh>

1. HOMOLOGAÇÃO

Criamos um projeto para realizar a preparação das bases de dados, antes do processo de cálculo, e também para a homologação dos dados:

* <https://github.com/DataViva/datavivaetl>

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. Com isso temos diversas tabelas com valores agregados e desagregados, com a mesma informação. Portanto foi necessário criar arauivos de conferência de valores na própria tabela e entre diferentes tabelas. Por exemplo, na tabela RAIS\_YB temos informações de salários e empregados por cidade e estado. Portanto o script deve somar os valores das cidade e comparar com o valor da cidade. O mesmo acontece entre tabelas, como por exemplo a RAIS\_YB e a RAIS\_YBI, sendo a primeira informando salário por cidade e a outra tabela por cidade e indústria. Portanto a soma dos salários das indústrias de uma cidade deve ser igual ao valor já agregado na tabela YB.

* + 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 um script 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 extração 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
* <https://github.com/DataViva/dataviva-scripts/blob/master/scripts/secex_new/sanity_check.py>
* <https://github.com/DataViva/dataviva-scripts/blob/master/scripts/secex_new/test_db_vars.py>

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;

* 1. Higher Education

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

* 1. School Census

Durante o processo de cálculo temos a possibilidade de realizar análises sobre os resultados encontrados. Isso ajuda a obter informações se o processo de cálculo foi realizado com sucesso ou não. Sendo assim temos scripts de exploração de dados, que imprimem na tela alguns diagnósticos sobre os resultados encontrados:

* <https://github.com/DataViva/dataviva-scripts/blob/master/scripts/sc/school_explore.py>
* <https://github.com/DataViva/dataviva-scripts/blob/master/scripts/sc/undergrad_explore.py>

Utilizando o mesmo conceito de visualizar os dados para realizar uma verificação, temos dois scripts que criam gráficos para demonstrar o comportamento e padrões dos dados.

* <https://github.com/DataViva/dataviva-scripts/blob/master/scripts/sc/school_plots.py>
* <https://github.com/DataViva/dataviva-scripts/blob/master/scripts/sc/undergrad_plots.py>
  1. Eletronic Invoice

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