Análises e Testes Comuns na auditoria - Uma abordagem em Python

O uso de Técnicas de Auditoria Assistida por Computador (CAATs do inglês) vem ganhando popularidade com os departamentos de auditoria, bem como com o clientes que eles servem. Embora alguns possam argumentar que o custo e o tempo envolvido na aquisição dessas técnicas não é justificável, ainda permanece verdadeiroa que estas fornecem valor acrescentado aos clientes porque eles apresentam uma imagem completa de um sistema e / ou transações da organização. Como resultado, um projeto de análise de dados deve provar os benefícios das CAATs tanto para a equipe de auditoria quanto aos clientes(Interno ou Externo).

Este trabalho visa retratar o demonstrado pelo autor Marcos F Silva na pagina https://sites.google.com/site/marcosfs2006/ onde ele utiliza a linguagem R para demonstrar a aplicação das técnicas mais comuns em auditoria utilizando-se da linguagem R.

No meu caso, vou utilizar a linguagem Python, que assim como o R, é uma linguagem open source.

Python é uma linguagem de programação de alto nível,interpretada, de script, imperativa, orientada a objetos, funcional, de tipagem dinâmica e forte. Foi lançada por Guido van Rossum em 1991. Atualmente possui um modelo de desenvolvimento comunitário, aberto e gerenciado pela organização sem fins lucrativos Python Software Foundation.

fonte: https://pt.wikipedia.org/wiki/Python

Técnicas básicas de Análise de dados

Apresentarei a seguir, como implementar no python algumas das técnicas de auditoria assistidas por computador mais comuns. São técnicas básicas que não fazem uso de métodos estatísticos mais sofisticados mas que na prática se mostram extremamente úteis. Esse trabalho não visa ser um tutorial sobre python, para isso, há um vasto material disponível na internet. Explicarei a função utilizada somente para os casos que julgar necessário para melhor compreensão do que foi apresentado.

Vamos importar todas as bibliotecas que iremos utilizar nesse trabalho.

In [1]:
```
import pandas as pd
import numpy as np
import datetime
```

1. Cálculo de novos campos

A criação de novos campos a partir de campos já existentes no conjunto de dados é uma tarefa bem simples. Vamos ilustrar este procedimento utilizando o que é talvez o pacote mais utilizado da linguagem python, o *pandas*. O pandas é uma biblioteca de software escrita para a linguagem de programação Python para manipulação e análise de dados. Em particular, oferece estruturas de dados e operações para manipular tabelas numéricas e séries temporais. Usaremos o conjunto de dados "invoices.csv" para ilustrar a criação de novos campos.

In [2]:
```
# Importar o conjunto de dados
invoices = pd.read_csv('invoices.csv', sep =";")
invoices.head()
```

Out[2]:

	Date	InvoiceNo	CustomerNo	SalesPerson	ProductNo	UnitPrice	Quantity	Amount
0	09/07/2003	20000	10220	8	8	9,20	41	377,20
1	21/08/2003	20001	10491	4	48	14,00	30	420,00
2	27/08/2003	20002	10704	3	43	15,00	25	375,00
3	28/05/2003	20003	10430	5	54	24,00	22	528,00
4	06/12/2003	20004	10841	17	11	15,00	21	315,00

In [3]:
```
invoices.dtypes # mostra o tipo de cada coluna do conjunto de dados
```

Out[3]:
```
Date           object
InvoiceNo       int64
CustomerNo      int64
SalesPerson     int64
ProductNo       int64
UnitPrice      object
Quantity        int64
```

```
Amount          object
dtype: object
```

Vemos aqui que as colunas 'UnitPrice' e 'Amount' são do tipo object do pandas, isso significa que ele esta entendendo os campos como 'string', texto, assim não será possível fazer operações matematicas que fazem parte dessa demonstração, para isso, iremos converter essas duas colunas.

In [4]:
```
invoices['UnitPrice'] = invoices['UnitPrice'].apply(lambda x: x.replace('$',
'').replace(',', '')).astype('float')
invoices['Amount'] = invoices['Amount'].apply(lambda x: x.replace('$', '').rep
lace(',', '')).astype('float')
invoices.dtypes
```

Out[4]:
```
Date            object
InvoiceNo        int64
CustomerNo       int64
SalesPerson      int64
ProductNo        int64
UnitPrice      float64
Quantity         int64
Amount         float64
dtype: object
```

Como vemos agora, os dois campos foram convertidos.

Vamos criar um novo campo na base de dados invoices chamado VlrAudit que será o produto dos campos Quant e PUnit.

In [5]:
```
invoices['VlrAudit'] = invoices['Quantity'] * invoices['UnitPrice']
invoices.head()
```

Out[5]:

	Date	InvoiceNo	CustomerNo	SalesPerson	ProductNo	UnitPrice	Quantity	Amount	Vlr
0	09/07/2003	20000	10220	8	8	920.0	41	37720.0	37
1	21/08/2003	20001	10491	4	48	1400.0	30	42000.0	42
2	27/08/2003	20002	10704	3	43	1500.0	25	37500.0	37
3	28/05/2003	20003	10430	5	54	2400.0	22	52800.0	52
4	06/12/2003	20004	10841	17	11	1500.0	21	31500.0	31

Agora vamos criar mais um campo que chamaremos de Dif e que será a diferença entre os campos Amount e VlrAudit.

In [6]:
```
invoices['Dif'] = invoices['Amount'] - invoices['VlrAudit']
invoices.head()
```

Out[6]:

	Date	InvoiceNo	CustomerNo	SalesPerson	ProductNo	UnitPrice	Quantity	Amount	Vlr
0	09/07/2003	20000	10220	8	8	920.0	41	37720.0	37
1	21/08/2003	20001	10491	4	48	1400.0	30	42000.0	42
2	27/08/2003	20002	10704	3	43	1500.0	25	37500.0	37
3	28/05/2003	20003	10430	5	54	2400.0	22	52800.0	52
4	06/12/2003	20004	10841	17	11	1500.0	21	31500.0	31

Agora vamos criar um novo campo chamado VlrAcum contendo o valor acumulado do campo Amount.

In [7]:
```
invoices['VlrAcum'] = invoices['Amount'].cumsum()
invoices.head()
```

Out[7]:

	Date	InvoiceNo	CustomerNo	SalesPerson	ProductNo	UnitPrice	Quantity	Amount	Vlr
0	09/07/2003	20000	10220	8	8	920.0	41	37720.0	37
1	21/08/2003	20001	10491	4	48	1400.0	30	42000.0	42
2	27/08/2003	20002	10704	3	43	1500.0	25	37500.0	37
3	28/05/2003	20003	10430	5	54	2400.0	22	52800.0	52
4	06/12/2003	20004	10841	17	11	1500.0	21	31500.0	31

Se quisermos deletar as colunas que foram acrescentadas e voltar ao conjunto de dados original, podemos fazer o seguinte:

In [8]:
```
invoices = invoices.drop(['VlrAudit','Dif','VlrAcum'],axis = 1) # O parâmetro
 axis = 1 indica que queremos que a coluna toda
#seja eliminada.
```

In [9]:
```
invoices.head()
```

Out[9]:

	Date	InvoiceNo	CustomerNo	SalesPerson	ProductNo	UnitPrice	Quantity	Amount
0	09/07/2003	20000	10220	8	8	920.0	41	37720.0
1	21/08/2003	20001	10491	4	48	1400.0	30	42000.0
2	27/08/2003	20002	10704	3	43	1500.0	25	37500.0
3	28/05/2003	20003	10430	5	54	2400.0	22	52800.0
4	06/12/2003	20004	10841	17	11	1500.0	21	31500.0

2. Adicionando novos registros

Algumas vezes surge a necessidade de adicionar novos registros (novas linhas) a um conjunto de dados. Esta situação surge, por exemplo, quando temos arquivos mensais contendo dados de faturamento de uma empresa, ou mesmo arquivos contendo as transações realizadas em determinada conta contábil e desejamos juntar estes arquivos para fazer a análise de todo o exercício financeiro. Esse exemplo será ilustrado com os conjuntos de dados Trans_Abril.xls e Trans_Maio.xls disponíveis no repositório.

In [10]:
```
trans_abril = pd.read_excel('Trans_Abril.xls')
trans_abril.head()
```

Out[10]:

	Númcartão	Valor	Data_Trans	Códigos	Númclien	Descrição
0	8590120032047834	270.63	2003-04-02	1731	1000	Contratos de eletricidade
1	8590120092563655	899.76	2003-04-02	1731	2000	Contratos de eletricidade
2	8590120233319873	730.46	2003-04-04	1750	250402	Contratos de carpintaria
3	8590120534914664	106.01	2003-04-08	1750	3000	Contratos de carpintaria
4	8590120674263418	309.37	2003-04-08	2741	1000	Publicações e impressões diversas

In [11]:
```
trans_maio1 = pd.read_excel('Trans_Maio.xls',sheet_name ='Trans1_Maio') # Tran
s_Maio possui duas abas, por isso temos que específicar no
                                                                        # parâmetro sheet_
name o nome da aba a ser importada.
trans_maio1.head()
```

Out[11]:

	Númcartão	Códigos	Data_Trans	Númclien	Descrição	Valor
0	8590 1252 7244 7003	4131	2003-05-27	925007	Linhas de ônibus, incluindo charters, ônibus d...	$108.01
1	8590128346463420	4214	2003-05-28	51593	Serviços de entrega - Local	$71.57
2	8590128263176714	4784	2003-05-29	503458	Tarifas telefônicas e pedágios	$5.83
3	8590128006917664	5992	2003-05-30	925007	Floristas	$152.97
4	8590 1294 0066 5510	4131	2003-03-31	51593	Linhas de ônibus, incluindo charters, ônibus d...	$390.33

In [12]:
```
trans_Maio2 = pd.read_excel('Trans_Maio.xls',sheet_name ='Trans2_Maio')
trans_Maio2.drop(columns = 'Unnamed: 5', axis = 1, inplace = True) # Planilha
 contém uma coluna sem informações
trans_Maio2.head()
```

Out[12]:

	Númcartão	Códigos	Data_Trans	Númclien	Descrição	Valor
0	8590-1224-9766-3807	2741	2003-05-04	962353	Publicações e impressões diversas	$510.43
1	8590122281964011	5021	2003-05-01	812465	Móveis de escritório e comerciais	$178.96
2	8590120784984566	3066	2003-05-02	51593	Southwest	$270.25
3	8590-1242-5362-1744	7922	2003-05-03	250402	Produtores teatrais (exceto filmes), agências ...	$182.62
4	8590125999743363	3007	2003-05-05	778088	Air France	$1031.70

In [13]:
```
trans_abril.shape # Shape demonstra quantas linhas e colunas há no conjunto de
dados. No caso há 281 linhas e 6 colunas.
```

Out[13]: (281, 6)

In [14]:
```
trans_maio1.shape
```

Out[14]: (86, 6)

In [15]:
```
trans_Maio2.shape
```

Out[15]: (114, 6)

Examinando os nomes das colunas nas três bases de dados, verifica-se que a base de abril possui

as colunas em posições diferentes das posições nas bases de maio1 e maio2. Mas isso não é um problema para o método append do pandas. Para ficar mais facil de visualizar, primeiro vamos juntar os 2 arquivos referentes a Maio, posteriormente, faremos um consolidado entre Abril e Maio.

In [16]:
```
trans_maio_total = trans_maio1.append(trans_Maio2, ignore_index= True)
trans_maio_total.shape
```

Out[16]:
```
(200, 6)
```

In [17]:
```
trans_maio_total.head()
```

Out[17]:

	Númcartão	Códigos	Data_Trans	Númclien	Descrição	Valor
0	8590 1252 7244 7003	4131	2003-05-27	925007	Linhas de ônibus, incluindo charters, ônibus d...	$108.01
1	8590128346463420	4214	2003-05-28	51593	Serviços de entrega - Local	$71.57
2	8590128263176714	4784	2003-05-29	503458	Tarifas telefônicas e pedágios	$5.83
3	8590128006917664	5992	2003-05-30	925007	Floristas	$152.97
4	8590 1294 0066 5510	4131	2003-03-31	51593	Linhas de ônibus, incluindo charters, ônibus d...	$390.33

In [18]:
```
consolidado = trans_abril.append(trans_maio_total, ignore_index=True, sort=False)
consolidado.head()
```

Out[18]:

	Númcartão	Valor	Data_Trans	Códigos	Númclien	Descrição
0	8590120032047834	270.63	2003-04-02	1731	1000	Contratos de eletricidade
1	8590120092563655	899.76	2003-04-02	1731	2000	Contratos de eletricidade
2	8590120233319873	730.46	2003-04-04	1750	250402	Contratos de carpintaria
3	8590120534914664	106.01	2003-04-08	1750	3000	Contratos de carpintaria
4	8590120674263418	309.37	2003-04-08	2741	1000	Publicações e impressões diversas

In [19]:
```
consolidado.shape
```

Out[19]:
```
(481, 6)
```

In [20]:
```
len(trans_abril) + len(trans_maio1) + len(trans_Maio2)
```

Out[20]:
```
481
```

Como podemos ver, a soma do numero de linhas das 3 tabelas bate com o consolidado.

3. Adicionando novos Campos

Para realizar esta tarefa o python dispõe da biblioteca *numpy*. Para ilustrar este procedimento serão criados os dados a serem utiliados. Serão criados 3 vetores que serão então combinados para formar um conjunto de dados único.

In [21]:
```
coluna1 = np.repeat(np.array(['Marcos','Maria','Carla']),[6]) # 18 elementos
coluna1
```

Out[21]:
```
array(['Marcos', 'Marcos', 'Marcos', 'Marcos', 'Marcos', 'Marcos',
       'Maria', 'Maria', 'Maria', 'Maria', 'Maria', 'Maria', 'Carla',
       'Carla', 'Carla', 'Carla', 'Carla', 'Carla'], dtype='<U6')
```

In [22]:
```
coluna2 = np.random.randn(18) # 18 números aleatórios provenientes de uma dist
ribuição normal
coluna2
```

Out[22]:
```
array([ 0.75160191,  0.15959898, -0.48275794,  0.66630409,  0.15705949,
        0.81462619, -1.06925119,  0.73345062,  1.01562002,  2.13847712,
        0.77974586,  1.09686934,  1.86790523,  1.10359197,  0.21471303,
       -0.36509248,  2.4998117 ,  0.50541895])
```

In [23]:
```
coluna3 = np.random.choice(["Jan", "Fev", "Mar", "Abr", "Mai", "Jun", "Jul",
"Ago", "Set", "Out", "Nov", "Dez"], replace= True,
                           size= 18)
coluna3
```

Out[23]:
```
array(['Jun', 'Nov', 'Set', 'Mar', 'Dez', 'Dez', 'Set', 'Ago', 'Jun',
       'Out', 'Abr', 'Nov', 'Abr', 'Mai', 'Mar', 'Mar', 'Out', 'Jul'],
      dtype='<U3')
```

In [24]:
```
# Juntando tudo em um DataFrame
Novo = pd.DataFrame({'Nome': coluna1,'Valor':coluna2,'Mês':coluna3})
Novo.head(10)
```

Out[24]:

	Nome	Valor	Mês
0	Marcos	0.751602	Jun
1	Marcos	0.159599	Nov
2	Marcos	-0.482758	Set
3	Marcos	0.666304	Mar
4	Marcos	0.157059	Dez
5	Marcos	0.814626	Dez
6	Maria	-1.069251	Set
7	Maria	0.733451	Ago
8	Maria	1.015620	Jun
9	Maria	2.138477	Out

4. Tabulação Cruzada

A tabulação cruzada consiste normalmente em se determinar a distribuição de frequências de uma, duas ou mais variáveis categóricas. O Python dispõe das funções crosstab() e groupby() que nos permitem implementar esta técnica.

In [25]:
```
# importar arquivo
rh_data = pd.read_excel('RH.xlsx')
rh_data.head()
```

Out[25]:

	Sexo	Estado Civil	Anos de estudo	Formação	Tempo de empresa	Unidade	Departamento	Cargo	Salário
0	Masculino	Casado	14	Sócio-econômicas	19	Curitiba	Produção	Assistente	16.67
1	Masculino	Viúvo	19	Sócio-econômicas	31	São Paulo	Vendas	Assistente	29.13
2	Feminino	Casado	18	Sócio-econômicas	28	Rio de Janeiro	Financeiro	Assi	21.8
3	Feminino	Casado	16	Sócio-econômicas	20	Rio de Janeiro	Vendas	Assistente	22.61
4	Masculino	Solteiro	15	Sócio-	15	Curitiba	Vendas	Auxiliar	16.67

econômicas

In [26]:
```
rh_data['Salário'] = pd.to_numeric(rh_data['Salário'],errors='coerce')
```

In [27]:
```
rh_data.dtypes
```

Out[27]:
```
Sexo                object
Estado Civil        object
Anos de estudo      object
Formação            object
Tempo de empresa    object
Unidade             object
Departamento        object
Cargo               object
Salário            float64
Bônus               object
dtype: object
```

In [28]:
```
rh_data.Sexo.value_counts()
```

Out[28]:
```
Masculino    2940
Feminino     2060
Name: Sexo, dtype: int64
```

Com os comandos acima, obteve-se a distribuição de frequência de uma única variável: Sexo. Caso se deseje obter a distribuição conjunta das variáveis Sexo e Estado.Civil, também conhecida como tabela de contingência, pode-se proceder da seguinte forma:

In [29]:
```
pd.crosstab(index= rh_data['Sexo'],columns= rh_data['Estado Civil'], margins= True,margins_name= 'Total')
```

Out[29]:

Estado Civil	Casado	Divorciado	Solteiro	Viúvo	Total
Sexo					
Feminino	1369	181	375	135	2060
Masculino	2080	238	424	198	2940
Total	3449	419	799	333	5000

Caso nosso interesse fosse totalizar os salários por Sexo, podemos proceder seguinte forma:

In [30]:
```
print(round(rh_data.groupby('Sexo', as_index = False)['Salário'].sum(),2))
```

```
        Sexo   Salário
0   Feminino  17871.31
1  Masculino  46206.43
```

5. Duplicidades

A identificação de valores duplicados em um conjunto de dados é uma tarefa executada rotineiramente pelos auditores com o objetivo de identificar repetições indevidas de valores. Para isso, o pacote Pandas possui uma função, a **duplicated**.

In [31]:
```
invoices = pd.read_csv('Invoices.csv',sep =";")
invoices.head()
```

Out[31]:

	Date	InvoiceNo	CustomerNo	SalesPerson	ProductNo	UnitPrice	Quantity	Amount
0	09/07/2003	20000	10220	8	8	9,20	41	377,20
1	21/08/2003	20001	10491	4	48	14,00	30	420,00
2	27/08/2003	20002	10704	3	43	15,00	25	375,00
3	28/05/2003	20003	10430	5	54	24,00	22	528,00
4	06/12/2003	20004	10841	17	11	15,00	21	315,00

In [32]:
```
# como primeiro argumento, pode-se passar uma ou mais colunas a serem usadas como base de comparação para procura
# de valores duplicados. No nosso  exemplo eu utilizei a coluna "InvoiceNo"(nº da Fatura).
invoices[invoices.duplicated('InvoiceNo', keep= False)]
```

Out[32]:

	Date	InvoiceNo	CustomerNo	SalesPerson	ProductNo	UnitPrice	Quantity	Amount
10	14/01/2003	20010	10439	19	38	7,45	28	208,60
11	14/01/2003	20010	10439	99	38	7,45	28	208,60

Como podemos ver, as linhas 10 e 11 contém valores duplicados.

6. Filtros

Ao se examinar um conjunto de dados, é frequente a situação em que o auditor tem interesse em selecionar para exame apenas um subconjunto do mesmo, ou seja, selecionar os registros que atendam a determinados critérios por ele definidos.

Assim o auditor pode desejar selecionar para análise, apenas os registros onde o valor das faturas, por exemplo, sejam superiores a determinado valor, ou faturas emitidas em um determinado período ou ainda registros relativos a funcionários de um determinado departamente de uma empresa. Para isso utilizaremos novamente o conjunto de dados RH.

In [33]:
```
 # Importando os dados
rh_data = pd.read_excel('RH.xlsx')
rh_data.head()
```

Out[33]:

	Sexo	Estado Civil	Anos de estudo	Formação	Tempo de empresa	Unidade	Departamento	Cargo	Salário
0	Masculino	Casado	14	Sócio-econômicas	19	Curitiba	Produção	Assistente	16.67
1	Masculino	Viúvo	19	Sócio-econômicas	31	São Paulo	Vendas	Assistente	29.13
2	Feminino	Casado	18	Sócio-econômicas	28	Rio de Janeiro	Financeiro	Assi	21.8
3	Feminino	Casado	16	Sócio-econômicas	20	Rio de Janeiro	Vendas	Assistente	22.61
4	Masculino	Solteiro	15	Sócio-econômicas	15	Curitiba	Vendas	Auxiliar	16.67

Supondo que se deseje selecionar apenas os registros relativos a funcionários do sexo feminino, pode-se fazer da seguinte forma:

In [34]:
```
rh_feminino = rh_data[rh_data['Sexo'] == 'Feminino' ]
rh_data['Salário'] = pd.to_numeric(rh_data['Salário'],errors='coerce')
rh_feminino.head()
```

Out[34]:

	Sexo	Estado Civil	Anos de estudo	Formação	Tempo de empresa	Unidade	Departamento	Cargo	Salá

2	Feminino	Casado	18	Sócio-econômicas	28	Rio de Janeiro	Financeiro	Assi	2
3	Feminino	Casado	16	Sócio-econômicas	20	Rio de Janeiro	Vendas	Assistente	22
9	Feminino	Casado	12	Humanas	16	Rio de Janeiro	Produção	Assistente	5
10	Feminino	Solteiro	13	Humanas	20	Florianópolis	Produção	Auxiliar	1
12	Feminino	Casado	13	Exatas	23	Rio de Janeiro	Pessoal	Assistente	5

Bem, e se quisermos filtrar os registros relativos a funcionários do sexo masculino, da unidade do Rio de Janeiro, do departamento de vendas e que possuam salário superior a $ 70,00?

In [35]:
```
rh_masculino = rh_data[(rh_data.Sexo  == 'Masculino')
                       & (rh_data.Unidade  == 'Rio de Janeiro')
                       & (rh_data.Departamento == 'Vendas')
                       & (rh_data.Salário >= 70.00)]
rh_masculino
```

Out[35]:

	Sexo	Estado Civil	Anos de estudo	Formação	Tempo de empresa	Unidade	Departamento	Cargo	Salá
367	Masculino	Viúvo	22	Sócio-econômicas	50	Rio de Janeiro	Vendas	Vice-Presidente	77
387	Masculino	Casado	23	Sócio-econômicas	42	Rio de Janeiro	Vendas	Gerente	88
397	Masculino	Casado	19	Sócio-econômicas	45	Rio de Janeiro	Vendas	Vice-Presidente	88
408	Masculino	Casado	28	Sócio-econômicas	37	Rio de Janeiro	Vendas	Vice-Presidente	141
607	Masculino	Casado	22	Sócio-econômicas	36	Rio de Janeiro	Vendas	Gerente	92
760	Masculino	Casado	19	Sócio-econômicas	41	Rio de Janeiro	Vendas	Gerente	75
798	Masculino	Casado	21	Sócio-econômicas	39	Rio de Janeiro	Vendas	Gerente	81
1316	Masculino	Casado	23	Sócio-econômicas	39	Rio de Janeiro	Vendas	Vice-Presidente	79

1435	Masculino	Casado	22	Sócio-econômicas	37	Rio de Janeiro	Vendas	Vice-Presidente	99
1567	Masculino	Casado	22	Sócio-econômicas	35	Rio de Janeiro	Vendas	Gerente	75
1696	Masculino	Casado	21	Sócio-econômicas	37	Rio de Janeiro	Vendas	Gerente	77
1827	Masculino	Casado	20	Sócio-econômicas	20	Rio de Janeiro	Vendas	Gerente	71
2591	Masculino	Casado	22	Sócio-econômicas	41	Rio de Janeiro	Vendas	Gerente	87
3013	Masculino	Casado	20	Sócio-econômicas	36	Rio de Janeiro	Vendas	Gerente	74
3229	Masculino	Casado	20	Sócio-econômicas	29	Rio de Janeiro	Vendas	Gerente	70
3520	Masculino	Casado	25	Sócio-econômicas	38	Rio de Janeiro	Vendas	Vice-Presidente	151
3696	Masculino	Casado	23	Sócio-econômicas	39	Rio de Janeiro	Vendas	Vice-Presidente	110
3777	Masculino	Casado	19	Sócio-econômicas	39	Rio de Janeiro	Vendas	Gerente	82
4230	Masculino	Casado	21	Sócio-econômicas	23	Rio de Janeiro	Vendas	Gerente	73
4550	Masculino	Casado	22	Sóci	30	Rio de Janeiro	Vendas	Gerente	74
4651	Masculino	Casado	23	Sócio-econômicas	48	Rio de Janeiro	Vendas	Gerente	92
4827	Masculino	Casado	24	Sócio-econômicas	36	Rio de Janeiro	Vendas	Vice-Presidente	72

7. Ordenação

A ordenação de um conjunto de dados com base nos valores de um determinado campo é uma tarefa comum quando se faz análise de dados. Normalmente o auditor busca, com este procedimento, identificar os maiores valores lançados, as maiores despesas, etc.

Utilizaremos o conjunto de dados Invoices.csv para exemplificar sua utilização na ordenação do conjunto de dados com base nos valores das faturas emitidas, o que permitirá identificar as de

maior valor.

In [36]:
```
invoices = pd.read_csv('Invoices.csv', sep =";")
invoices.head()
```

Out[36]:

	Date	InvoiceNo	CustomerNo	SalesPerson	ProductNo	UnitPrice	Quantity	Amount
0	09/07/2003	20000	10220	8	8	9,20	41	377,20
1	21/08/2003	20001	10491	4	48	14,00	30	420,00
2	27/08/2003	20002	10704	3	43	15,00	25	375,00
3	28/05/2003	20003	10430	5	54	24,00	22	528,00
4	06/12/2003	20004	10841	17	11	15,00	21	315,00

A ordenação dessa base de dados com base nos valores das faturas (campo Amount) pode ser feita de forma crescente ou decrescente como ilustrado a seguir:

In [37]:
```
invoices.sort_values('Amount').head(10) # Por padrão, a função sort_values ord
ena por ordem crescente de valores.
```

Out[37]:

	Date	InvoiceNo	CustomerNo	SalesPerson	ProductNo	UnitPrice	Quantity	Amount
1342	13/05/2003	21343	10224	10	34	2,50	4	10,00
3617	16/09/2003	23618	10265	9	39	10,00	10	100,00
4823	14/10/2003	24824	10454	1	68	12,50	8	100,00
2638	21/09/2003	22639	10998	12	5	12,50	8	100,00
551	11/12/2003	20552	10042	12	5	12,50	8	100,00
4436	25/09/2003	24437	10526	12	5	12,50	8	100,00
4565	29/10/2003	24566	10586	14	5	12,50	8	100,00
975	27/01/2003	20976	10712	22	50	10,00	10	100,00
4244	10/08/2003	24245	10058	17	72	7,75	13	100,75
363	28/10/2003	20364	10954	7	72	7,75	13	100,75

Se o intuito for ordenar de forma descrescente, devemos proceder da seguinte maneira:

In [38]:
```
invoices.sort_values('Amount', ascending= False).head(10)
```

Out[38]:

	Date	InvoiceNo	CustomerNo	SalesPerson	ProductNo	UnitPrice	Quantity	Amount
4298	16/09/2003	24299	10937	2	24	23,25	43	999,75
1239	30/12/2003	21240	10989	21	24	23,25	43	999,75
2170	13/02/2003	22171	10771	7	15	123,79	8	990,32
1624	19/12/2003	21625	10153	15	15	123,79	8	990,32
2431	14/05/2003	22432	10830	22	15	123,79	8	990,32
3737	10/03/2003	23738	10894	11	15	123,79	8	990,32
806	02/12/2003	20807	10408	13	56	30,00	33	990,00
3450	29/04/2003	23451	10640	13	59	55,00	18	990,00
244	08/09/2003	20245	10607	15	56	30,00	33	990,00
643	17/04/2003	20644	10802	3	19	30,00	33	990,00

Ordenamento com base em duas colunas

Caso nosso interesse fosse em ordenar a base de dados com base na coluna SalesPerson (ordem crescente) e na coluna Amount (ordem decrescente), podemos fazer da seguinte forma:

In [39]:
```
invoices.sort_values(['SalesPerson','Amount'],axis =0,
                     ascending= [True,False], inplace = True)
invoices.head()
```

Out[39]:

	Date	InvoiceNo	CustomerNo	SalesPerson	ProductNo	UnitPrice	Quantity	Amount
3988	27/10/2003	23989	10559	1	73	49,30	20	986,00
676	26/01/2003	20677	10866	1	32	12,00	8	96,00
782	04/04/2003	20783	10335	1	76	9,50	10	95,00
4250	19/07/2003	24251	10580	1	70	19,00	5	95,00
2566	07/01/2003	22567	10371	1	61	31,00	3	93,00

Como podemos ver, a coluna SalesPerson esta ordenada de forma crescente, enquanto a coluna Amount de forma decrescente.

8. *Gaps*

Diversos documentos são emitidos seguindo uma numeração sequencial; o que permite um maior controle sobre os mesmos. Exemplos disso são a numeração sequencial das notas fiscais emitidas por uma empresa, dos cheques emitidos, das notas de empenhos, dos contratos celebrados, etc.

Um recurso muito útil e usualmente disponível nos softwares de auditoria de uso geral é a possibilidade de idenficar a existência de itens faltantes em um conjunto de dados, ou seja, verificar se existe algum gap (lacuna) na numeração sequencial.

Vamos ilustrar como fazer isso usando o conjunto de dados Invoices.csv.

A abordagem utilizada será descrita a seguir:

In [40]:
```
# Importar arquivos
invoices = pd.read_csv('Invoices.csv', sep =";")
invoices.head()
```

Out[40]:

	Date	InvoiceNo	CustomerNo	SalesPerson	ProductNo	UnitPrice	Quantity	Amount
0	09/07/2003	20000	10220	8	8	9,20	41	377,20
1	21/08/2003	20001	10491	4	48	14,00	30	420,00
2	27/08/2003	20002	10704	3	43	15,00	25	375,00
3	28/05/2003	20003	10430	5	54	24,00	22	528,00
4	06/12/2003	20004	10841	17	11	15,00	21	315,00

O conjunto de dados será ordenado com base na numeração sequencial das faturas e cada elemento será subtraído do seu antecessor. Se não houver falha na numeração sequencial, espera-se obter como resultado um vetor contendo apenas 1. Onde o resultado for diferente de 1 haverá uma falha. Vejamos como fazer isso na prática.

1. • Ordenação do conjunto de dados com base no número das faturas

In [41]:
```
faturas = invoices.sort_values('InvoiceNo')
faturas.head()
```

Out[41]:

	Date	InvoiceNo	CustomerNo	SalesPerson	ProductNo	UnitPrice	Quantity	Amount

0	09/07/2003	20000	10220	8	8	9,20	41	377,20
1	21/08/2003	20001	10491	4	48	14,00	30	420,00
2	27/08/2003	20002	10704	3	43	15,00	25	375,00
3	28/05/2003	20003	10430	5	54	24,00	22	528,00
4	06/12/2003	20004	10841	17	11	15,00	21	315,00

1. • Cálculo das diferenças

In [42]:
```
faturas['differ'] = faturas.InvoiceNo.diff()
faturas.head()
```

Out[42]:

	Date	InvoiceNo	CustomerNo	SalesPerson	ProductNo	UnitPrice	Quantity	Amount	diff
0	09/07/2003	20000	10220	8	8	9,20	41	377,20	Na
1	21/08/2003	20001	10491	4	48	14,00	30	420,00	1
2	27/08/2003	20002	10704	3	43	15,00	25	375,00	1
3	28/05/2003	20003	10430	5	54	24,00	22	528,00	1
4	06/12/2003	20004	10841	17	11	15,00	21	315,00	1

1. • Identificação dos registros onde há gaps

In [43]:
```
diferencas = faturas[faturas['differ']!= 1.0].iloc[1:] #iloc aqui é para elimi
narmos a primeira linha, pois o resultado dela é NaN,
# já que não é possível ubtrairmos a primeira linha pela linha antecessora.
diferencas
```

Out[43]:

	Date	InvoiceNo	CustomerNo	SalesPerson	ProductNo	UnitPrice	Quantity	Amount	di
11	14/01/2003	20010	10439	99	38	7,45	28	208,60	
12	29/08/2003	20012	10919	11	31	21,05	28	589,40	
23	17/01/2003	20024	10459	24	61	31,00	42	1302,00	

In [44]:
```
faturas.iloc[[10,11,12,22,23],:]
```

Out[44]:

	Date	InvoiceNo	CustomerNo	SalesPerson	ProductNo	UnitPrice	Quantity	Amount	di
10	14/01/2003	20010	10439	19	38	7,45	28	208,60	
11	14/01/2003	20010	10439	99	38	7,45	28	208,60	
12	29/08/2003	20012	10919	11	31	21,05	28	589,40	
22	29/11/2003	20022	10355	15	12	46,00	12	552,00	
23	17/01/2003	20024	10459	24	61	31,00	42	1302,00	

Como podemos ver, as faturas nº 20010 estão duplicadas, falta a fatura nº 20011 e a 20023.

9. Estratificação

A estratificação consiste em classificar os registros de um arquivo de dados em estratos mutuamente exclusivos. Em auditoria, esta classificação é feita usualmente com base em valores monetários. O procedimento consiste em se estabelecer faixas de valores e indicar, para cada registro, a que faixa de valor ele pertence.

Para ilustrar a execução deste procedimento, será utilizado o conjunto de dados Invoices.csv.

O primeiro passo para se definir os estratos consiste em se estabelecer as faixas de valores para a variável a ser utilizada na estratificação nas quais se deseja classificar os registros. Para isso, faz-se necessário conhecer os valores máximo e mínimo do conjunto de dados, o que pode ser feito facilmente com a função min () e max().

In [45]:
```
# Importar arquivos
invoices = pd.read_csv('Invoices.csv', sep =";")
invoices.head()
```

Out[45]:

	Date	InvoiceNo	CustomerNo	SalesPerson	ProductNo	UnitPrice	Quantity	Amount
0	09/07/2003	20000	10220	8	8	9,20	41	377,20
1	21/08/2003	20001	10491	4	48	14,00	30	420,00
2	27/08/2003	20002	10704	3	43	15,00	25	375,00
3	28/05/2003	20003	10430	5	54	24,00	22	528,00
4	06/12/2003	20004	10841	17	11	15,00	21	315,00

In [46]:
```
# Transformando o campo "Amount" e "UnitPrice" para numérico.
invoices['UnitPrice'] = invoices['UnitPrice'].apply(lambda x: x.replace('$',
'').replace(',', '.')).astype('float')
invoices['Amount'] = invoices['Amount'].apply(lambda x: x.replace('$', '').rep
lace(',', '.')).astype('float')
```

In [47]:
```
print('O menor valor Total de fatura é: ', invoices.Amount.min())
```

```
O menor valor Total de fatura é:  5.0
```

In [48]:
```
print('O maior valor Total de fatura é: ', invoices.Amount.max())
```

```
O maior valor Total de fatura é:  13438.5
```

Considerando que os valores das faturas variam entre 5,00 e 13.438,50, desejamos estabelecer 3 estratos (baixos valores, valores medianos e altos valores) poderíamos ter os seguintes estratos:

Estrato 1 - 5,00 a 1.000,00

Estrato 2 - 1.000,01 a 10.000,00

Estrato 3 - 10.000,01 a 13.438,50

A definição da quantidade de estratos (3 neste exemplo) e da amplitude de cada faixa é uma escolha subjetiva do auditor. Deve-se observar que as faixas de valores podem ou não ter a mesma amplitude.

Para realizar a classificação dos registros da base de dados em cada um dos estratos definidos utiliza-sa a função cut(), como exemplificado a seguir:

In [49]:
```
invoices['Estratos'] = pd.cut(invoices.Amount,[0.00, 1000.00, 10000.00, 13500.
00],
                              labels = ['Baixo','Médio','Alto'])

invoices.head()
```

Out[49]:

	Date	InvoiceNo	CustomerNo	SalesPerson	ProductNo	UnitPrice	Quantity	Amount	Es
0	09/07/2003	20000	10220	8	8	9.2	41	377.2	
1	21/08/2003	20001	10491	4	48	14.0	30	420.0	
2	27/08/2003	20002	10704	3	43	15.0	25	375.0	

3	28/05/2003	20003	10430	5	54	24.0	22	528.0
4	06/12/2003	20004	10841	17	11	15.0	21	315.0

10. Sumarização

É relativamente freqüente a situação em que o auditor deseja obter um resumo dos dados (valor mínimo, valor máximo, média, medianas, totais, etc.) com base nos registros correspondentes a subconjuntos dos dados em exame. Estes subconjuntos normalmente são definidos pelos valores assumidos por variáveis categóricas.

Vejamos um exemplo.

O conjunto de dados Invoices.csv, contém informações sobre o faturamento de uma empresa em todo o exercício de 2003. Dentre os campos existentes no conjunto de dados há o campo Date, que contém a data em que a nota fiscal foi emitida, a partir do qual podemos facilmente criar um novo campo na base de dados contendo o mês de emissão da NF.

Observe que este campo terá doze valores distintos ("Jan", "Fev", ..., "Dez") e cada mês define um subconjunto dos dados, para os quais podemos estar interessados em calcular a média, o total , o valor mínimo, o valor máximo, a mediana, etc. dos valores faturados (coluna Amount).

No caso específico do conjunto de dados Invoices.csv, pode ser de interesse do auditor verificar o faturamento médio em cada um dos doze meses do exercício financeiro ou qualquer outra medida resumo para cada mês.

In [50]:
```
invoices = pd.read_csv('Invoices.csv', sep =";")
invoices.head()
```

Out[50]:

	Date	InvoiceNo	CustomerNo	SalesPerson	ProductNo	UnitPrice	Quantity	Amount
0	09/07/2003	20000	10220	8	8	9,20	41	377,20
1	21/08/2003	20001	10491	4	48	14,00	30	420,00
2	27/08/2003	20002	10704	3	43	15,00	25	375,00
3	28/05/2003	20003	10430	5	54	24,00	22	528,00
4	06/12/2003	20004	10841	17	11	15,00	21	315,00

In [51]:
```
invoices.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4999 entries, 0 to 4998
Data columns (total 8 columns):
Date           4999 non-null object
InvoiceNo      4999 non-null int64
CustomerNo     4999 non-null int64
SalesPerson    4999 non-null int64
ProductNo      4999 non-null int64
UnitPrice      4999 non-null object
Quantity       4999 non-null int64
Amount         4999 non-null object
dtypes: int64(5), object(3)
memory usage: 253.9+ KB
```

In [52]:
```
invoices['Date'] = pd.to_datetime(invoices.Date) # Converter o campo Date para data
# Transformando o campo "Amount" e "UnitPrice" para numérico.
invoices['UnitPrice'] = invoices['UnitPrice'].apply(lambda x: x.replace('$', '').replace(',', '.')).astype('float')
invoices['Amount'] = invoices['Amount'].apply(lambda x: x.replace('$', '').replace(',', '.')).astype('float')
```

In [53]:
```
invoices['Mes'] = pd.DatetimeIndex(invoices['Date']).month_name(locale = 'portuguese')
```

In [54]:
```
invoices.head()
```

Out[54]:

	Date	InvoiceNo	CustomerNo	SalesPerson	ProductNo	UnitPrice	Quantity	Amount	Me
0	2003-09-07	20000	10220	8	8	9.2	41	377.2	Setembr
1	2003-08-21	20001	10491	4	48	14.0	30	420.0	Agost
2	2003-08-27	20002	10704	3	43	15.0	25	375.0	Agost
3	2003-05-28	20003	10430	5	54	24.0	22	528.0	Mai
4	2003-06-12	20004	10841	17	11	15.0	21	315.0	Junh

Cálculo do faturamento médio mensal usando a função

groupby()

In [55]:
```
agregado = invoices.groupby('Mes')['Amount'].mean()
```

In [56]:
```
idx = ['Janeiro', 'Fevereiro', 'Março', 'Abril',
       'Maio', 'Junho', 'Julho', 'Agosto',
       'Setembro', 'Outubro', 'Novembro', 'Dezembro']
```

In [57]:
```
agregado.index = idx
```

In [58]:
```
print(agregado)
```

```
Janeiro      784.417044
Fevereiro    751.299372
Março        799.364848
Abril        753.461481
Maio         857.524030
Junho        740.119382
Julho        879.050913
Agosto       877.123108
Setembro     718.753643
Outubro      760.167678
Novembro     727.223156
Dezembro     781.173012
Name: Amount, dtype: float64
```

Estatísticas mensais

In [59]:
```
estatisticas = invoices.groupby('Mes')['Amount'].describe()
estatisticas.index = idx
estatisticas
```

Out[59]:

	count	mean	std	min	25%	50%	75%	max
Janeiro	406.0	784.417044	1332.651098	15.5	228.8875	422.50	938.7	13438.5
Fevereiro	446.0	751.299372	992.000988	5.0	228.1250	468.60	877.0	9222.5
Março	462.0	799.364848	1386.994542	7.0	217.0000	413.25	798.0	12911.5
Abril	378.0	753.461481	1152.756103	18.0	199.6250	470.00	837.0	13438.5
Maio	402.0	857.524030	1482.590352	7.5	204.7800	433.00	796.5	10803.5

Junho	421.0	740.119382	1180.501529	15.5	220.0000	425.00	790.2	12384.5
Julho	416.0	879.050913	1592.778040	5.0	251.3125	484.15	878.5	12648.0
Agosto	415.0	877.123108	1580.633090	10.0	238.0000	456.00	847.5	12384.5
Setembro	409.0	718.753643	1045.735970	5.0	216.0000	462.00	779.0	11330.5
Outubro	379.0	760.167678	1162.504486	9.5	228.2750	475.00	879.8	12911.5
Novembro	450.0	727.223156	1198.454666	7.0	209.2500	450.00	786.6	10540.0
Dezembro	415.0	781.173012	1256.659617	9.0	209.5000	437.50	838.1	11330.5

11. Combinar Arquivos (merging)

Em determinadas situações ocorre de os dados necessários à execução de uma tarefa estarem em arquivos diferentes e é necessário reuní-los em um único conjunto de dados.

Para exemplificar, considere o caso em que o auditor necessite confirmar com terceiros os valores que compõem o saldo da conta Clientes, por exemplo.

Os dados cadastrais, como endereço, código de identificação do cliente, CEP, etc, constam de um arquivo e os valores devidos pelos clientes, em outro. Ocorre que o auditor necessita do endereço e dos valores devidos para elaborar uma carta de circularização. Para isso, é indispensável que em ambos os arquivos exista um campo que identifique o cliente de forma única(nesse caso utilizaremos a coluna "account"). Para a execução deste procedimento o python dispõe da função pandas.merge(), cuja utilização será mostrada mais adiante.

Estes dois conjuntos de dados são arquivos texto de formato fixo e, dessa forma, sua importação depende de conhecermos o layout dos arquivos, o que consta do documento Descricao Arquivos Dados_v3.doc que acompanha os conjuntos de dados utilizados.

Com base nas informações contidas naquele documento, a importação dos dados pode ser feita da seguinte forma:

In [60]:
```
# importar o arquivo de contas a receber
contas_a_receber = pd.read_fwf('Arfile.ASC', sep ="\t", widths=[11, 4, 4, 15,
8],
                               header = None,
                               names = ['account', 'division', 'store', 'balan
ce', 'duedate'])
```

In [61]:
```
contas_a_receber.head()
```

Out[61]:

	account	division	store	balance	duedate
0	S0000309077	246	20	13192.42	20010101
1	S0000041943	87	3	260.97	20010103
2	S0000143191	87	20	9541.28	20010106
3	S0000459709	9045	20	2254.19	20010110
4	S0000030187	139	4	2286.84	20010110

In [62]:
```
# Carregar o arquivo de endereço dos clientes
endereco = pd.read_fwf('Address.ASC', sep= '\t', widths= [11, 33, 33, 30, 25, 5],
                       header = None,
                       names = ['account', 'name1', 'name2', 'street', 'cityst', 'zip'])
```

In [63]:
```
endereco.head()
```

Out[63]:

	account	name1	name2	street	cityst	zip
0	S0000031637	LYNDA RANSEGNOLA	NaN	1 CHARLOTTE LANE	DENVILLE, AZ	72134
1	S0000249225	SOPHIE F. NATHAN	NaN	1 COOLIDGE ST.	NORWALK, CT	6850
2	S0000032500	MERLE DEL POLITO	NaN	1 EAST SHAWNEE TRAIL	WHARTON, AZ	72185
3	S0000800468	KERRI STRACCO	NaN	1 FRANCIS TERRACE P. O. BOX 68	STANHOPE, AZ	72174
4	S0000001037	JULIE ANN LAMPE	NaN	1 NEWHAMPSHIRE STREET	NEWTON, AZ	72160

In [64]:
```
novo_conjunto = pd.merge(contas_a_receber, endereco[['account','name1', 'name2', 'street', 'cityst','zip']],
                         on= 'account', how='inner')
```

In [65]:
```
novo_conjunto.head()
```

Out[65]:

	account	division	store	balance	duedate	name1	name2	street

0	S0000309077	246	20	13192.42	20010101	HYACINTHE H. NUBER	NaN	66 CLEARMONT AVENUE	D
1	S0000041943	87	3	260.97	20010103	ITF JENNIFER ZANIEWSKI	PAULA ZANIEWSKI	6 COBB STREET	RC
2	S0000143191	87	20	9541.28	20010106	BETTY KEMMERER	NaN	30 BELLOWS LANE	
3	S0000459709	9045	20	2254.19	20010110	PETER BAYSA	NaN	18 CRESTWOOD RD.	RC
4	S0000030187	139	4	2286.84	20010110	JOSEPHINE PORFIDO	NaN	16 LAKEWOOD DRIVE	D

No procedimento realizado acima, também conhecido como inner join, o conjunto de dados resultante (no nosso caso novo_conjunto) conterá apenas as informações relativas às contas que existirem em ambos os cojuntos de dados. As contas que estiverem no conjunto de dados contas_receber e não estiverem em endereco não constarão da base de dados resultante. O mesmo ocorrendo ao contrário, ou seja, as contas que estiverem no conjunto de dados endereco mas não estiverem em contas_receber também não serão apresentados. O inner join mostrará a interseção das duas bases.