JPNB1_Fornecedores

September 4, 2023

ANÁLISE DAS CONTRATAÇÕES NO ESTADO DE SANTA CATARINA:

JPNB 01 - ETL DOS DADOS DE FORNECEDORES

Autor: Maurício Vasconcellos Leão Lyrio, Dr. | Página Oficial: www.vll.adm.br

---

1 Instalação das bibliotecas

[ ]:
```
# Manipulação de dados
import pandas as pd
import numpy as np

# Interação com bancos de dados
# SQLite
import sqlite3
# MySQL
#import mysql.connector
# MongoDB
#from pymongo import MongoClient

# Visualização de dados
#import matplotlib.pyplot as plt
#import seaborn as sns

# Ignorar warnings
import warnings
warnings.filterwarnings('ignore')

# Versões dos pacotes utilizados neste Jupyter notebook
#!pip install -q -U watermark
```

```
%reload_ext watermark
%watermark -a "Mauricio Vasconcellos Leão Lyrio | vll.adm.br" --iversions
```

---

2 Carregamento dos datasets

Para iniciar nosso processo de análise de dados iremos **carregar o dataset com os dados cadastrais de fornecedores** recebido por meio de solicitação via LAI. Esse procedimento carrega o dataset do arquivo .csv original e o armazena em um dataframe pandas denominado ***df***, para que possamos manipulá-lo posteriormente.

[ ]:
```
# Carregando o dataset de fornecedores
df = pd.read_csv('datasets/ELIC_fornecedores_cadastro.csv')
```

---

3 Análise exploratória dos dados

Após carregar o dataset damos início ao processo de análise exploratório, buscando **analisar a qualidade e integridade dos dados**. O processo de análise exploratório nos ajuda a ter uma visão geral do dataset e que tipo de pré-processamento precisaremos realizar nos dados a fim de deixá-los prontos para as etapas posteriores.

[ ]:
```
# Verificando se o dataset foi carregado corretamente e seu tipo
type(df)
```

[ ]:
```
# Verificando o formato do dataframe
df.shape
```

[ ]:
```
# Listando as colunas do dataframe
df.columns
```

[ ]:
```
# Listando as informações gerais do dataframe
df.info()
```

Até o momento conseguimos carregar os dados, verificar o tamanho do dataframe, suas colunas, o tipo de dado de cada coluna e se existem valores ausentes. Pelo info do dataframe é possível perceber que existem campos com valores nulos. Precisaremos definir o que fazer com esses campos e que tipo de tratamento iremos dar para os valores nulos, voltaremos a discutir essa questão na fase do pré-processamento de dados.

Pelo info do dataframe também é possível perceber que quase todos os campos são não-numéricos. O único campo numérico é o ***cnpj***, que, na verdade, não é uma variável quantitativa e sim um código que representa o cadastro de pessoa jurídica do fornecedor. Posteriormente iremos ajustar o tipo de dado desse campo, por hora iremos somente descrever os demais campos de nosso dataset e visualizar uma amostra dos dados.

[ ]:
```
# Descrevendo os dados não-numéricos
df.describe(include=object)
```

[ ]:
```
df.head()
```

Com a visualização de uma amostra do dataset finalizamos a análise exploratória. Outros tipos de análise poderiam ser feitos nessa fase, porém, para nosso objetivo de preparar o dataset o que vimos até agora é suficiente. Passemos então à próxima fase do processo, o pré-processamento dos dados.

---

4 Pré-processamento

Na fase de análise exploratória identificamos que nosso dataset possui campos nulos e também que um dos campos foi definido de forma equivocada como numérico. Vamos agora tratar esses problemas e também analisar a necessidade de outros tipos de transformação de dados. Comecemos com a limpeza dos dados.

4.1 Limpeza

Conforme visto anteriormente, nosso dataset possui uma série de campos com valores nulos. Vamos analisar melhor essa situação e definir o que fazer com esses valores. para isso criaremos uma nova **tabela com a distribuição percentual de valores nulos por coluna**.

[ ]:
```
# Criando uma lista vazia para armazenar as informações de nome e tipo de␣
 ↪coluna.
colunas_info = []

# Iterando pelas colunas do dataframe
for coluna in df.columns:
    coluna_nome = coluna
    coluna_tipo = df[coluna].dtype
    coluna_nulos = df[coluna].isnull().sum()
    coluna_nulos_perc = (coluna_nulos/len(df))*100
    colunas_info.
 ↪append((coluna_nome,coluna_tipo,coluna_nulos,coluna_nulos_perc))

# Criando um novo dataframe e exibindo as informações das colunas
df_colunas_info = pd.DataFrame(colunas_info, columns=['Coluna','Tipo','Q Nulo',␣
 ↪'% Nulo'])
print(df_colunas_info)
```

Com a nova tabela fica mais fácil evidenciar os valores ausentes do dataframe. No caso, as colunas ***cidade***, ***uf***, ***pais*** e ***produtos_habilitados*** apresentam valores ausentes. Em projetos de data-science, em geral, utiliza-se como regra para tratamento de valores ausentes as seguintes opções:

- Para valores ausentes $>= 50\%$, descartamos a variável;
- Para valores ausentes $< 50\%$, tratar os valores ausentes;

- Para valores ausentes $< 2\%$, descartar os valores ausentes.

Apesar dessa regra geral, é importante analisar o dataframe e verificar a forma mais adequada para tratamento dos valores ausentes e, principalmente, justificar as escolhas feitas no decorrer do tratamento dos dados. Em nosso caso, **como os valores ausentes são menos de 2% dos valores dos campos vamos excluí-los**. Porém, vale salientar que, ao excluir os registros cujos campos estão ausentes perdemos parte da informação no dataset, vale refletir sobre a relevância da perda dessa informação.

Nesse caso, como os registros no dataset estão com granularidade definida em nível de produtos habilitados, acredita-se que os dados básicos dos fornecedores não serão perdidos devido à exclusão desses registros.

[ ]:
```
# Criando uma cópia do dataframe original e excluindo os registros com valores nulos.
df1 = df.dropna()
```

[ ]:
```
df1.isnull().sum()
```

Temos agora um novo dataframe ***df1*** com os registros com valore ausentes excluídos. Mantivemos em memória o dataset original para o caso de queremos retornar à essa versão em outro momento. Passemos agora à tranformação dos dados.

4.2 Transformação

Iniciaremos nossa fase de transformação de dados ajustando o tipo de dado referente à coluna ***cnpj*** do fornecedor. Conforme verificamos anteriormente, essa coluna foi definida como numérica, do tipo int64. Porém, por se tratar de um código de referência de cada fornecedor, ela na verdade é categórica.

[ ]:
```
# Alterando o tipo de dado da coluna cnpj
df1['cnpj'] = df1['cnpj'].astype(object)
```

[ ]:
```
# Listando as informações gerais para conferir a alteração
df1['cnpj'].info()
```

Conforme evidenciado acima, agora todos os campos de nosso dataset são campos categóricos e poderemos utilizar os métodos de string para a coluna. Dando sequência á transformação de dados, analisando a coluna ***produtos_habilitados*** percebemos que essa coluna apresenta a informação dos produtos concatenando duas informações, o grupo e a classe do produto.

Esse tipo de conhecimento é o que chamamos de conhecimento de negócio, ou seja, para perceber esses nuances nos dados o analista precisa conhecer pelo menos um pouco da área de negócio sobre a qual está trabalhando, dessa forma a possibilidade de ter insights úteis em relação ao tópico aumenta, aprimorando as possibilidades de análise.

Iremos separar a coluna de ***produtos_habilitados*** em duas colunas, a primeira para armazenar os grupos de produtos e a segunda para armazenar a classe dos produtos.

[ ]:
```
# Criando as novas colunas e separando os registros da coluna original
```

```
df1[['Grupo','Classe','Descrição']] = df1['produtos_habilitados'].str.split(' - ',2,expand=True)
df1.head()
```

[ ]:
```
# Dividindo a coluna ***Grupo*** em duas para separar os códigos de grupo e de classe.
df1['Grupo1'] = df1['Grupo'].str[0:2]
df1['Grupo2'] = df1['Grupo'].str[2:]

# Renomeando a coluna de grupo/classe original
df1.rename(columns={'Grupo':'Código GC'},inplace=True)

# Criando as colunas de descrição de grupo e de classe
df1['Grupo_desc'] = df1['Grupo1']+' - '+df1['Classe']
df1['Classe_desc'] = df1['Grupo2']+' - '+df1['Descrição']
df1.head()
```

[ ]:
```
# Excluindo as colunas que não serão necessárias
df1.drop(columns=['Classe','Descrição','Grupo1','Grupo2'],inplace=True)
df1.head()
```

[ ]:
```
# Ajustando os nomes das novas colunas
df1.rename(columns={'Grupo_desc':'Grupo','Classe_desc':'Classe'},inplace=True)
```

[ ]:
```
df1.isnull().sum()
```

Ao gerar o info do novo dataset percebemos que ao transformar o dataset original acabamos por gerar uma nova coluna que contém valores nulos. Vamos agora analisar a situação e decidir o que fazer com os dados inconsistentes.

[ ]:
```
# Descrevendo os dados não-numéricos
df1.describe(include=object)
```

[ ]:
```
# Comparando a descrição dos dataframes
describe_df = df.describe(include=object)
describe_df1 = df1.describe(include=object)

# Adicionando os nomes dos dataframes como índices
describe_df.index=['df'] * len(describe_df)
describe_df1.index=['df1'] * len(describe_df)

# Concatenando os resultados em um único dataframe
df_comparacao = pd.concat([describe_df,describe_df1])

# Listando a tabela resultante
df_comparacao
```

Comparando a descrição do ***df1*** com a descrição do ***df*** percebemos que no processo de limpeza

e transformação de dados foram perdidos registros de fornecedores (o df original tinha 20.121 cnpjs únicos e o novo df1 possui 19.194). Conforme dito anteriormente, a decisão de manter ou não determinadas informações do dataset é do analista, porém, é importante ter em conta as justificativas para as escolhas feitas no decorrer do processo. No caso, optaremos por manter o dataframe com os registros de *Classe* nulos, dado que já havíamos perdido informações de 927 fornecedores na primeira etapa do processo de limpeza.

Porém, para que o dataframe se torne mais "amigável", vamos substituir os campos nos quais existem valores nulos por um valor categórico para fins de uso na etapa de análise.

[ ]:
```
# Preenchendo valores nulos da categoria Classe
df1['Classe'].fillna('Classe não definida',inplace=True)

# Verificando o resultado
df1.isnull().sum()
```

Com a finalização do processo de tranformação de dados nosso dataframe está pronto para ser carregado em banco de dados ou exportado em formatos de arquivos para análise posterior. É o que iremos fazer agora, na fazer de geração de dados de saída.

---

5 Geração de dados de saída (Data output)

Uma vez finalizada a fase de limpeza e transformação, agora iremos dar saída ao dataset gerado para fins de análise. Faremos isso em forma de arquivos e em registros em banco de dados.

5.1 Gravação em arquivos

[ ]:
```
# Gravando em formato .csv
df1.to_csv('datasets/df1_fornecedores.csv',index=False)
```

[ ]:
```
# Gravando em formato .json
df1.to_json('datasets/df1_fornecedores.json')
```

[ ]:
```
# Gravando em formato .xls
df1.to_excel('datasets/df1_fornecedores.xlsx',index=False)
```

5.2 Gravação em DB relacional

Gravando em banco de dados SQLite

[ ]:
```
# Importando a biblioteca para interação com o SQLite
#import sqlite3

# Criando uma conexão ao banco de dados SQLite
cnn=sqlite3.connect('database/da12023db.db')

# Copiando nosso dataframe para o banco de dados
```

```
df1.to_sql('Fornecedores',cnn)
```

Abaixo o código para gravação em banco de dados ***MySQL***, não executaremos esse script porque necessitamos ter SGBD instalado.

[ ]:
```
# Importando a biblioteca para interação com o MySQL
#import mysql.connector

# Configurando os parâmetros da conexão
#config={
#    'user':'seu usuário';
#    'password':'sua_senha';
#    'host':'localhost';    # ou endereço do servidor MySQL
#    'database':'seu_bando_de_dados'
#}

# Criando uma conexão ao banco de dados
#try:
#    conn=mysql.connector.connect(**config)
#    if conn.is_connected():
#        print('Conexão ao banco de dados bem sucedida')
#except mysql.connector.Error as err:
#    print(f'Erro ao conectar ao banco de dados: {err}')

# Copiando nosso dataframe para o banco de dados

# Fechando a conexão (ao terminar de utilizar)
#conn.close()
```

5.3 Gravação em datalake (BD não-relacional)

Abaixo o código para gravação em banco de dados ***MongoDB***, não executaremos esse script porque necessitamos ter SGBD instalado.

[ ]:
```
# Importando a biblioteca para interação com o MongoDB
#from pymongo import MongoClient

# Configurando os parâmetros de conexão
#client=MongoClient('mongodb://localhost:27017/')

# Acessando um banco de dados específico
#db=client['appdb']

# Criando uma coleção no banco de dados
#collection = db['Fornecedores']

# Carregando o dataframe
#data = df1.to_dict(orient='records')
```

```
# Inserindo os registros no MongoDB
#collection.insert_many(data)

# Listando as coleções disponíveis
#print(db.list_collection_names())
```

[ ]:
```
#fornecedores=collection.find()
#for fornecedor in fornecedores:
#    print(fornecedor)
```

[ ]:
```
# Fechando a conexão ao MongoDB
#client.close()
```

Com a geração dos dados de saída, nosso trabalho de ***ETL*** de dados terminou. Fizemos a limpeza, transformação e carga de dados para arquivos de saíde e bancos de dados relacionais e não-relacionais. Agora passaremos à etapa de visualização de dados (dataviz) que faremos utilizando o Microsoft Power BI como ferramenta.

FIM

[ ]:
```
# Versão da linguagem Python e arquitetura do Jupyter Notebook
import platform
print('Versão da linguagem Python utilizada neste notebook:', platform.
 ↪python_version())
print('Arquitetura do Jupyter utilizada neste notebook:', platform.
 ↪architecture()[0])
```