JPNB2_Contratos

September 4, 2023

ANÁLISE DAS CONTRATAÇÕES NO ESTADO DE SANTA CATARINA:

JPNB 02 - ETL DOS DADOS DE CONTRATOS

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

# Visualização de dados
# import matplotlib.pyplot as plt
# import seaborn as sns
# !pip install -q plotly
# import plotly.express as px
# import plotly.graph_objects as go
# from plotly.subplots import make_subplots

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

# Estatística
# import scipy
```

```
# from scipy.stats import normaltest
# from scipy.stats import chi2_contingency

# Engenharia de atributos
# from sklearn.pipeline import Pipeline
# from sklearn.impute import SimpleImputer
# from sklearn.preprocessing import OneHotEncoder, labelEncoder, OrdinalEncoder
# from sklearn.compose import ColumTransformer
# !pip install -q category_encoders
# import category_encoders as ce

# 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

Os dados de contratos estabelecidos pelo Estado de Santa Catarina foram obtidos do Portal de Dados Abertos do Estado de Santa Catarina. Vamos utilizar dois datasets referentes aos contratos firmados pelo Estado:

- Arquivo em formato .csv abrangendo o período entre 2011 e 2021;
- Arquivo em formato .json abrangendo o ano de 2022.

Em seguida iremos consolidar esses datasets em um único para que possamos fazer nosso trabalho de análise.

2.1 Carregando os contratos de 2011 a 2021

[ ]:
```
# Carregando o dataset de contratos de 2011 a 2021 a partir do Portal de Dados␣
 ↪Abertos (dados.sc.gov.br)
df_2021 = pd.read_csv('https://dados.sc.gov.br/dataset/
 ↪93dab950-e805-4388-8418-cfb3b73f1623/resource/
 ↪ac64ba57-bac8-4969-9248-cb9c9b76415d/download/contratos-2011-2021.csv')
```

[ ]:
```
# Verificando se o dataset foi carregado corretamente e seu tipo
type(df_2021)
```

[ ]:
```
# Verificando o formato do dataframe
df_2021.shape
```

[ ]:
```
# Listando as colunas do dataframe
df_2021.columns
```

2.2 Carregando os contratos de 2022

[ ]:
```
# Carregando o dataset de contratos de 2022 a partir do Portal de Dados Abertos␣
↪(dados.sc.gov.br)
df_2022 = pd.read_json('https://dados.sc.gov.br/dataset/
↪93dab950-e805-4388-8418-cfb3b73f1623/resource/
↪d4a78b11-fd97-4114-8f8b-2bf45ffab0be/download/contratos-2022.json')
```

[ ]:
```
# Verificando se o dataset foi carregado corretamente e seu tipo
type(df_2022)
```

[ ]:
```
# Verificando o formato do dataframe
df_2022.shape
```

Opa! o dataset de contratos de 2022 possui duas colunas a mais que o de 2011 a 2021. Qual será a diferença? Vamos verificar ...

[ ]:
```
# Listando as colunas do dataframe
df_2022.columns
```

Evindenciamos que o **df_2022** possui duas colunas a mais ('IDCONTRATADOMASCARADO', 'CDCREDOR'). Vamos ver o que que é o conteúdo dessas colunas.

[ ]:
```
df_2022[['IDCONTRATADO','IDCONTRATADOMASCARADO','CONTRATADO','CDCREDOR']].head()
```

A coluna 'IDCONTRATADOMASCARADO' é somente o CNPJ do contratado com a máscara respectiva (que insere os caracteres especiais). A coluna 'CDCREDOR' precisamos confirmar no dicionário de dados o que significa. Lembre-se, o dicionário de dados é seu amigo!

2.3 Consolidando os dataframes

No caso do nosso projeto, mesmo sabendo que existem duas colunas a mais no dataframe com os dados de contratos de 2022, iremos concatenar as tabelas para ter uma tabela única que utilizaremos nas próximas fases do trabalho.

[ ]:
```
# Concatenando os dois datasets verticalmente em um único dataframe consolidado
df1 = pd.concat([df_2021,df_2022], ignore_index=True)
```

---

3 Análise exploratória dos dados

Agora que temos um dataframe no qual os dados dos contratos estão consolidados, daremos sequência ao processo de análise, neste ponto iremos verificar a qualidade e integridade dos dados disponíveis.

[ ]:
```
# Verificando se o dataframe foi carregado corretamente e seu tipo
type(df1)
```

[ ]:
```
# Verificando o formato do dataframe
df1.shape
```

[ ]:
```
df1.info()
```

[ ]:
```
# Definindo o formato de exibição com duas casas decimais e separação de milhar
def format_float(value):
    return '{:.2f}'.format(value)
# Aplicar a função de formatação ao dataframe
pd.options.display.float_format= format_float

# Descrevendo os dados numéricos (selecionadas apenas as colunas que realmente␣
 ↪se constituem em dados numéricos)
df1[['VLORIGINAL','VLATUAL','NUPRAZO','VLGARANTIA','VLPERCGARANTIA','VLADITADO','DIASORIGINAIS
 ↪describe()
```

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

---

4 Pré-processamento

Na fase de análise exploratória identificamos que nosso dataframe possui diversos campos nulos e campos que apesar de estar com tipo numérico, parecem ser informações categóricas. Iremos agora analisar essas questões.

4.1 Limpeza

Conforme visto anteriormente, nosso dataframe 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 df1.columns:
    coluna_nome = coluna
    coluna_tipo = df1[coluna].dtype
    coluna_nulos = df1[coluna].isnull().sum()
    coluna_nulos_perc = (coluna_nulos/len(df1))*100
    colunas_info.
 ↪append((coluna_nome,coluna_tipo,coluna_nulos,coluna_nulos_perc))
```

```
# Criando um novo dataframe e exibindo as informações das colunas
df1_colunas_info = pd.DataFrame(colunas_info, columns=['Coluna','Tipo','Q␣
 ↪Nulo', '% Nulo'])
print(df1_colunas_info)
```

Com a nova tabela fica mais fácil evidenciar os valores ausentes do dataframe. Em projetos de datascience, 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.

Primeiro vamos excluir os registros cujos valores são nulos e representam menos de 2% dos registros da coluna ['DTINICIO','DTFIM','DTFIMATUAL','NUPROCESSO']

[ ]:
```
df2 = df1.copy()
```

[ ]:
```
df2 = df2.dropna(subset=['DTINICIO'])
```

[ ]:
```
df2.isnull().sum()
```

Nesse primeiro drop de registros excluímos 162 contratos que não possuiam data de início cadastrada. Ao excluir os registros sem data de início, acabamos por excluir também os contratos que não possuíam data fim e data fim atualizada, que eram os mesmos contratos. Vamos excluir agora os contratos cuja coluna 'NUPROCESSO' esteja vazia. Nesse procedimento iremos perder registros de mais 2.254 contratos.

[ ]:
```
df2 = df2.dropna(subset=['NUPROCESSO'])
```

Agora vamos excluir as colunas cujos registros nulos sejam mais de 50% de registros da coluna. Manteremos as colunas que estão com 88.28% de valores nulos dado que a primeira vista se constiem nos contratos de obras advindos do sistema SICOP, caso dropemos essas colunas perderemos a informação referente a isso.

[ ]:
```
df2 = df2.
 ↪drop(columns=['NUPROCESSOFORMATADO','TAGS','NMBEMPUBLICO','NMREGIMEEXECUCAO','DEMULTA','NUA
```

[ ]:
```
df2.isnull().sum()
```

4.2 Transformação

Iniciaremos a etapa de transformação de dados fazendo o preenchimento dos campos com registros nulos em nosso dataframe.

[ ]:
```
# Preenchendo os valores nulos da coluna 'RESUMO' com o texto 'Resumo não␣
 ↪apresentado'
df2['RESUMO'].fillna('Resumo não apresentado', inplace=True)
```

[ ]:
```
# Criando uma lista com os demais campos com valores nulos
colunas_com_nulos = df2.columns[df2.isnull().any()].tolist()
print(colunas_com_nulos)
```

[ ]:
```
# Alterando o tipo de dado das colunas com valores nulos para object
df2[colunas_com_nulos] = df2[colunas_com_nulos].astype(object)

# Preenchendo os valores nulos com o termo 'Não definido'
df2.fillna('Não definido',inplace=True)
```

[ ]:
```
df2.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
df2.to_csv('datasets/df2_contratos.csv',index=False)
```

[ ]:
```
# Gravando em formato .json
df2.to_json('datasets/df2_contratos.json')
```

[ ]:
```
# Gravando em formato .xls
df2.to_excel('datasets/df2_contratos.xlsx',index=False)
```

5.2 Gravação em DB relacional

Gravando em banco de dados SQLite

[ ]:
```
# A biblioteca sqlite3 foi importada no começo do projeto

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

# Copiando nosso dataframe para o banco de dados
df2.to_sql('Contratos',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])
```