INSTITUTO INFNET

ESCOLA SUPERIOR DE TECNOLOGIA

GRADUAÇÃO EM ANÁLISE E DESENVOLVIMENTO

PYTHON PARA DADOS

TESTE DE PERFORMANCE – TP3

ALUNO: GABRIEL GOMES DE SOUZA

PROFESSOR(A): DÁCIO MOREIRA DE SOUZA

E-MAIL: gabriel.gsouza@al.infnet.edu.br

Sumário

1. Conteúdo

Parte 01

Questão 01

```python
# Utilizando o SQLAlchemy, crie uma engine para conexão com um banco de dados sqlite de sua escolha, carregue ele para um DataFrame Pandas e exporte as informações para dois arquivos json diferentes, cada um com uma orientação.

# pip install sqlalchemy pandas
import sqlite3
from sqlalchemy import create_engine
import pandas as pd

PATH = 'Python para Dados\\TP03\\Parte 1\\'
DB_PATH = PATH + 'ex01_banco.db'
PATH_OUTPUT_RECORDS = PATH + 'ex01_data_records.json'
PATH_OUTPUT_SPLIT = PATH + 'ex01_data_split.json'

# def criarBanco():
#     # Criação do banco de dados
#     conn = sqlite3.connect(DB_PATH)
#     cursor = conn.cursor()

#     # Criação da tabela 'data'
#     cursor.execute('''
#     CREATE TABLE IF NOT EXISTS data (
#         id INTEGER PRIMARY KEY,
#         nome TEXT,
#         idade INTEGER,
#         cidade TEXT
#     )
#     ''')

#     # dados de exemplo
```

```
#     cursor.executemany('''
#     INSERT INTO data (nome, idade, cidade)
#     VALUES (?, ?, ?)
#     ''', [
#         ('Ana', 23, 'São Paulo'),
#         ('Bruno', 34, 'Rio de Janeiro'),
#         ('Carlos', 45, 'Belo Horizonte'),
#         ('Daniela', 29, 'Curitiba'),
#         ('Eduardo', 39, 'Porto Alegre')
#     ])
#     conn.commit()
#     conn.close()

def exportar_dados():
    """
    Cria uma conexão com um banco de dados SQLite, cria um DataFrame com os dados da tabela 'data'
    e exporta para dois arquivos JSON com orientações diferentes (records e split). Caso ocorra algum erro durante o processo de exportação, a exceção será capturada e
    uma mensagem de erro é exibida.
    """
    try:
        # Criação da engine de conexão com o banco de dados SQLite
        engine = create_engine(f'sqlite:///{DB_PATH}')

        # Carregamento dos dados da tabela 'data' para um DataFrame
        df = pd.read_sql_table('data', con=engine)

        # Exportação dos dados para arquivo JSON com orientação 'records'
        df.to_json(PATH_OUTPUT_RECORDS, orient='records', indent=4)
```

```
        # Exportação dos dados para arquivo JSON com
orientação 'split'
        df.to_json(PATH_OUTPUT_SPLIT, orient='split',
indent=4)

        print("Exportação concluída com sucesso.")
    except Exception as e:
        print(f"Ocorreu um erro: {e}")


# criarBanco()
exportar_dados()
```

Questão 02

```
# Crie um DataFrame com os seguintes dados do tipo:
{'nome': ['Ana', 'Carlos'], 'idade': [25, 30], 'cidade':
['Rio de Janeiro', 'São Paulo']}. Conecte-se a um banco
de dados SQLite (pessoas.db) e armazene o DataFrame em
uma tabela chamada pessoas.

import pandas as pd
import sqlite3

PATH = 'Python para Dados\\TP03\\Parte 1\\'
DATABASE_PATH = PATH + 'ex02_pessoas.db'

data = {
    'nome': ['Ana', 'Carlos'],
    'idade': [25, 30],
    'cidade': ['Rio de Janeiro', 'São Paulo']
}
df = pd.DataFrame(data)

# Conecta no banco de dados SQLite (ou criar se não
```

```
existir)
conn = sqlite3.connect(DATABASE_PATH)

# Armazena o DataFrame na tabela pessoas
df.to_sql('pessoas', conn, if_exists='replace',
index=False)
conn.close()

print("Dados armazenados na tabela 'pessoas' do banco de
dados 'ex02_pessoas.db'")
```

Questão 03

```
# Use o arquivo csv "dadoSujoTP3.csv" do repositório
compartilhado com a turma, carregue-o para um DataFrame e
proceda o tratamento de dados inicial, visualizando o
dados, realizando uma análise exploratória básica para
identificar dados vazios, inconsistentes ou com tipo
errado. Realize a limpeza dos dados, adicione duas
colunas geradas a partir dos dados existentes e exporte o
DataFrame para um arquivo excel.

import pandas as pd

PATH = 'Python para Dados\\TP03\\Parte
1\\dadoSujoTP3.csv'
df = pd.read_csv(PATH)

# análise exploratória básica:
print("Primeiras linhas do DataFrame:")
print(df.head())

print("\nInformações sobre o DataFrame:")
print(df.info())

print("\nValores nulos por coluna:")
```

```
print(df.isnull().sum())

# padronização das datas
df['data_venda'] = pd.to_datetime(df['data_venda'],
errors='coerce').dt.strftime('%Y-%m-%d')
df['data_inscricao'] =
pd.to_datetime(df['data_inscricao'],
errors='coerce').dt.strftime('%Y-%m-%d')

# correção de emails inválidos
df['email'] = df['email'].str.replace('example..com',
'example.com')
df['email'] = df['email'].apply(lambda x: x if
pd.notnull(x) and '@' in x else
'email.desconhecido@example.com')

# remoção de duplicatas
df = df.drop_duplicates()

# substitui todos os valores NaN (valores ausentes) na
coluna idade pela média calculada dessa coluna.
df['idade'] = df['idade'].fillna(df['idade'].mean())

# transforma idade em int
df['idade'] = df['idade'].astype(int)

# correção de preços e total de vendas (substitui os
valores 100.000,00 em total_venda por 10.000,00.)
df.loc[df['total_venda'] == 100000.00, 'total_venda'] =
10000.00

# substitui todos os valores NaN (valores ausentes) na
coluna preco_unitario por 0.
df['preco_unitario'] = df['preco_unitario'].fillna(0)

# remover telefone inválido
```

```
df['telefone'] = df['telefone'].fillna('N/A')

# salvar o CSV limpo:
OUTPUT_PATH_EXCEL = 'Python para Dados\\TP03\\Parte 1\\dadoLimpoTP3.xlsx'
df.to_excel(OUTPUT_PATH_EXCEL, index=False)

print(f"\nDataFrame limpo exportado para {OUTPUT_PATH_EXCEL}")
```

Parte 02

Questão 01

```
# Escreva uma função que solicita ao usuário dois números e tenta dividir o primeiro pelo segundo. Use tratamento de exceções para lidar com os possíveis erros como: a entrada inválida (ex: texto ao invés de número), conversão de tipo, a divisão por zero e outros que você considere pertinente, informando ao usuário a natureza do erro e lidando com cada de uma forma diferente.

def dividir_numeros():
    """
    Solicita 2 números e tenta dividir o 1º pelo 2º, com tratamento de exceções:
    - ValueError: Informar o usuário que a entrada deve ser numérica.
    - ZeroDivisionError: Informar o usuário que a divisão por zero não é permitida.
    - Exception: Informar o usuário de qualquer outro erro inesperado que possa ocorrer.

    A função continua solicitando a entrada até que uma divisão válida seja realizada.
```

```
    """
    while True:
        try:
            num1 = input("Digite o primeiro número: ")
            num2 = input("Digite o segundo número: ")
            num1 = float(num1)
            num2 = float(num2)

            resultado = num1 / num2

            print(f"O resultado da divisão de {num1} por
{num2} é: {resultado}")
            break

        except ValueError:
            print("Erro: Você digitou um valor inválido.
Por favor, digite apenas números.")

        except ZeroDivisionError:
            print("Erro: Divisão por zero não é
permitida. Por favor, digite um segundo número diferente
de zero.")

        except Exception as e:
            print(f"Ocorreu um erro inesperado: {e}")
```

Questão 02

```
# Crie uma função que recebe uma lista de números e
retorna uma nova lista apenas com os números pares e
positivos. Use uma classe de exceção personalizada para
tratar o caso em que um número negativo seja encontrado,
interrompendo a execução e retornando a lista acumulada
até o ponto da exceção.
```

```python
class ErroNumeroNegativo(Exception):
    def __init__(self, message="Número negativo encontrado."):
        self.message = message
        super().__init__(self.message)

def filtrar_pares_positivos(lista):
    """
    Filtra uma lista de números para retornar apenas os números pares e positivos.

    Args:
        lista: Lista contendo números inteiros.

    Returns:
        lista: Lista contendo apenas os números pares e positivos da lista de entrada.

    Raises:
        ErroNumeroNegativo: Se um número negativo for encontrado na lista, uma exceção é lançada indicando o número negativo encontrado e a execução é interrompida.
    """
    numeros_filtrados = []
    for num in lista:
        if num < 0:
            raise ErroNumeroNegativo(f"Número negativo encontrado: {num}")
        elif num % 2 == 0:
            numeros_filtrados.append(num)
    return numeros_filtrados
```

```
try:
    lista_numeros = [1, 2, 3, -4, 5, 6, -7, 8, 9, 10]
    resultado = filtrar_pares_positivos(lista_numeros)
    print("Lista filtrada:", resultado)
except ErroNumeroNegativo as e:
    print("Erro:", e)
```

Questão 03

```
# Escreva um código que tenta abrir e ler um arquivo
chamado config.yaml. Se o arquivo não for encontrado,
crie-o com conteúdo padrão (config: default) e exiba uma
mensagem informando a criação do arquivo.

import yaml # pip install pyyaml
import os

def ler_configuracao():
    """
    Tenta abrir e ler um arquivo chamado "config.yaml",
se o arquivo não for encontrado, ele é criado com um
conteúdo padrão ("config: default")
    """
    arquivo_config = 'config.yaml'
    PATH = "Python para Dados\\TP03\\Parte 2\\" +
arquivo_config

    if os.path.exists(PATH):
        with open(PATH, 'r') as file:
            try:
                configuracoes = yaml.safe_load(file)
                print("Configurações encontradas:",
configuracoes)
            except yaml.YAMLError as e:
```

```
                print(f"Erro ao ler o arquivo {PATH}:
{e}")
    else:
        # se o arquivo não existir é criado com um
conteúdo padrão
        configuracoes = {'config': 'default'}
        with open(PATH, 'w') as file:
            yaml.safe_dump(configuracoes, file)
        print(f"Arquivo {arquivo_config} criado com
sucesso.")
ler_configuracao()
```