

PREENCHA SEU NOME COMPLETO AQUI:

MBA em Ciência de Dados

Redes Neurais e Arquiteturas Profundas

Análise de Dados com Base em Processamento Massivo em Paralelo

Prova Final

Material Produzido por:

Profa. Dra. Cristina Dutra de Aguiar

Prof. Dr. Moacir A. Ponti

CEMEAI - ICMC/USP São Carlos

A prova final contém 1 questão, dividida em 3 itens. Por favor, procurem por Questão para encontrar a especificação da questão e por RESOLVER para encontrar a especificação do item a ser solucionado. Também é possível localizar a questão e os itens utilizando o menu de navegação.

O notebook contém a constelação de fatos da BI Solutions que deve ser utilizada para responder à questão e também todas as bibliotecas, bases de dados, inicializações, instalações, importações, geração de dataFrames, geração de visões temporárias e conversão dos tipos de dados necessárias para a realização da questão.

INSTRUÇÕES:

1) Você deve exportar esse notebook com sua solução para as questões da prova em formato .py e fazer upload no Moodle. Atenção: você não deve fazer upload de um arquivo notebook (.ipynb), mas sim um arquivo texto .py contendo os códigos python que utilizou para resolver as questões. O arquivo .py pode ser gerado através da opção:

File --> Download as --> Python (.py) disponível no Jupyter Notebook.

ou File --> Download .py no Google Colab

Caso não esteja utilizando o Jupyter, copie e cole seu código em um arquivo ASCII (Texto) salvando com a extensão .py

2) Você deve salvar esse notebook com sua solução para as questões da prova em formato .pdf e fazer upload no Moodle

3) Os arquivos devem ser nomeados com seu nome e sobrenome, sem espaços. Exemplo: moacirponti.py e moacirponti.pdf

4) É OBRIGATÓRIO conter no cabeçalho (início) do arquivo um comentário / texto com o seu nome completo

Desejamos uma boa prova!

1 Constelação de Fatos da BI Solutions

A aplicação de *data warehousing* da BI Solutions utiliza como base uma contelação de fatos, conforme descrita a seguir.

Tabelas de dimensão

- data (dataPK, dataCompleta, dataDia, dataMes, dataBimestre, dataTrimestre, dataSemestre, dataAno)
- funcionario (funcPK, funcMatricula, funcNome, funcSexo, funcDataNascimento, funcDiaNascimento, funcMesNascimento, funcAnoNascimento, funcCidade, funcEstadoNome, funcEstadoSigla, funcRegiaoNome, funcRegiaoSigla, funcPaisNome, funcPaisSigla)
- equipe (equipePK, equipeNome, filialNome, filialCidade, filialEstadoNome, filialEstadoSigla, filialRegiaoNome, filialRegiaoSigla, filialPaisNome, filialPaisSigla)
- cargo (cargoPK, cargoNome, cargoRegimeTrabalho, cargoEscolaridadeMinima, cargoNivel)
- cliente (clientePK, clienteNomeFantasia, clienteSetor, clienteCidade, clienteEstadoNome, clienteEstadoSigla, clienteRegiaoNome, clienteRegiaoSigla, clientePaisNome, clientePaisSigla)

Tabelas de fatos

- pagamento (dataPK, funcPK, equipePK, cargoPK, salario, quantidadeLancamentos)
- negociacao (dataPK, equipePK, clientePK, receita, quantidadeNegociacoes)

▼ 2 Configurações

▼ 2.1 Obtenção dos Dados da BI Solutions

```
#instalando o módulo wget
%%capture
!pip install -q wget
!mkdir data
```

```
#baixando os dados das tabelas de dimensão e das tabelas de fatos
import wget
```

```
url = "https://raw.githubusercontent.com/cdaciferri/DataMartBISolutions/main/data.csv"
wget.download(url, "data/data.csv")
```

```
url = "https://raw.githubusercontent.com/cdaciferri/DataMartBISolutions/main/funcio
wget.download(url, "data/funcionario.csv")

url = "https://raw.githubusercontent.com/cdaciferri/DataMartBISolutions/main/equipe
wget.download(url, "data/equipe.csv")

url = "https://raw.githubusercontent.com/cdaciferri/DataMartBISolutions/main/cargo.
wget.download(url, "data/cargo.csv")

url = "https://raw.githubusercontent.com/cdaciferri/DataMartBISolutions/main/client
wget.download(url, "data/cliente.csv")

url = "https://raw.githubusercontent.com/cdaciferri/DataMartBISolutions/main/pagame
wget.download(url, "data/pagamento.csv")

url = "https://raw.githubusercontent.com/cdaciferri/DataMartBISolutions/main/negoci
wget.download(url, "data/negociacao.csv")
```

▼ 2.2 Instalações e Inicializações

```
#instalando Java Runtime Environment (JRE) versão 8
%%capture
!apt-get remove openjdk*
!apt-get update --fix-missing
!apt-get install openjdk-8-jdk-headless -qq > /dev/null

#baixando Apache Spark versão 3.0.0
%%capture
!wget -q https://archive.apache.org/dist/spark/spark-3.0.0/spark-3.0.0-bin-hadoop2.
!tar xf spark-3.0.0-bin-hadoop2.7.tgz && rm spark-3.0.0-bin-hadoop2.7.tgz

import os
#configurando a variável de ambiente JAVA_HOME
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
#configurando a variável de ambiente SPARK_HOME
os.environ["SPARK_HOME"] = "/content/spark-3.0.0-bin-hadoop2.7"

%%capture
#instalando o pacote findspark
!pip install -q findspark==1.4.2
#instalando o pacote pyspark
!pip install -q pyspark==3.0.0
```

▼ 2.3 Bibliotecas

```
import findspark
findspark.init()
```

```

from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("pyspark-notebook").master("local[*]").getOrCreate()

from pyspark.sql.types import IntegerType
from pyspark.sql.types import FloatType
from pyspark.sql.functions import round, desc

import random
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
from numpy.random import seed
from tensorflow.random import set_seed
from tensorflow import keras
from tensorflow.keras import layers

```

▼ 2.4 Geração dos DataFrames em Pandas da BI Solutions

Nesta seção são gerados os DataFrames em Pandas. Atenção aos nomes desses DataFrames.

```

pd.set_option('display.float_format', lambda x: '%.2f' % x)

cargoPandas = pd.read_csv('https://raw.githubusercontent.com/cdaciferri/DataMartBI/refs/heads/main/data/cargo.csv')
clientePandas = pd.read_csv('https://raw.githubusercontent.com/cdaciferri/DataMartBI/refs/heads/main/data/cliente.csv')
dataPandas = pd.read_csv('https://raw.githubusercontent.com/cdaciferri/DataMartBI/refs/heads/main/data/data.csv')
equipePandas = pd.read_csv('https://raw.githubusercontent.com/cdaciferri/DataMartBI/refs/heads/main/data/equipe.csv')
funcionarioPandas = pd.read_csv('https://raw.githubusercontent.com/cdaciferri/DataMartBI/refs/heads/main/data/funcionario.csv')
negociacaoPandas = pd.read_csv('https://raw.githubusercontent.com/cdaciferri/DataMartBI/refs/heads/main/data/negociacao.csv')
pagamentoPandas = pd.read_csv('https://raw.githubusercontent.com/cdaciferri/DataMartBI/refs/heads/main/data/pagamento.csv')

```

▼ 2.5 Geração dos DataFrames em Spark da BI Solutions

Nesta seção são gerados dos DataFrames em Spark. Atenção aos nomes desses DataFrames.

```

#criando os DataFrames em Spark
cargo = spark.read.csv(path="data/cargo.csv", header=True, sep=",")
cliente = spark.read.csv(path="data/cliente.csv", header=True, sep=",")
data = spark.read.csv(path="data/data.csv", header=True, sep=",")
equipe = spark.read.csv(path="data/equipe.csv", header=True, sep=",")
funcionario = spark.read.csv(path="data/funcionario.csv", header=True, sep=",")
negociacao = spark.read.csv(path="data/negociacao.csv", header=True, sep=",")
pagamento = spark.read.csv(path="data/pagamento.csv", header=True, sep=",")

#convertendo os dados necessários para o tipo de dado inteiro
colunas_cargo = ["cargoPK"]
colunas_cliente = ["clientePK"]
colunas_data = ["dataPK", "dataDia", "dataMes", "dataBimestre", "dataTrimestre", "dataAno"]

```

```

colunas_equipe = ["equipePK"]
colunas_funcionario = ["funcPK", "funcDiaNascimento", "funcMesNascimento", "funcAnc
colunas_negociacao = ["equipePK", "clientePK", "dataPK", "quantidadeNegociacoes"]
colunas_pagamento = ["funcPK", "equipePK", "dataPK", "cargoPK", "quantidadeLancamer

for coluna in colunas_cargo:
    cargo = cargo.withColumn(coluna, cargo[coluna].cast(IntegerType()))

for coluna in colunas_cliente:
    cliente = cliente.withColumn(coluna, cliente[coluna].cast(IntegerType()))

for coluna in colunas_data:
    data = data.withColumn(coluna, data[coluna].cast(IntegerType()))

for coluna in colunas_equipe:
    equipe = equipe.withColumn(coluna, equipe[coluna].cast(IntegerType()))

for coluna in colunas_funcionario:
    funcionario = funcionario.withColumn(coluna, funcionario[coluna].cast(IntegerType

for coluna in colunas_negociacao:
    negociacao = negociacao.withColumn(coluna, negociacao[coluna].cast(IntegerType())

for coluna in colunas_pagamento:
    pagamento = pagamento.withColumn(coluna, pagamento[coluna].cast(IntegerType()))

#convertendo os dados necessários para o tipo de dado float
colunas_negociacao = ["receita"]
colunas_pagamento = ["salario"]

for coluna in colunas_negociacao:
    negociacao = negociacao.withColumn(coluna, negociacao[coluna].cast(FloatType()))

for coluna in colunas_pagamento:
    pagamento = pagamento.withColumn(coluna, pagamento[coluna].cast(FloatType()))

#criando as visões temporárias
cargo.createOrReplaceTempView("cargo")
cliente.createOrReplaceTempView("cliente")
data.createOrReplaceTempView("data")
equipe.createOrReplaceTempView("equipe")
funcionario.createOrReplaceTempView("funcionario")
negociacao.createOrReplaceTempView("negociacao")
pagamento.createOrReplaceTempView("pagamento")

```

▼ 3 Questão

A empresa BI Solutions está realizando uma investigação baseada no projeto e treinamento de uma rede neural, usando como base seus dados históricos mantidos na Constelação de Fatos. O modelo resultante deve ser usado para obter uma predição, a qual é voltada à análise da investigação esperada.

IMPORTANTE: Leia a questão com muita atenção, desde que vários passos da questão já se encontram implementados. Os locais nos quais os comandos de resposta para os itens da questão devem ser especificados são identificados em comentários.

▼ 3.1 Investigação dos Dados Históricos

A primeira parte para solucionar a investigação consiste na obtenção de dados históricos de interesse. Isso deve ser feito por meio da especificação de uma consulta OLAP, segundo as instruções detalhadas a seguir.

▼ 3.1.1 Instruções para a Especificação da Consulta OLAP

- A **consulta OLAP** pode ser especificada usando qualquer uma das três opções a seguir (escolha SOMENTE UMA forma)
 - Usando **Pandas** (conceitos apresentados na Aula 05).
 - Usando a **linguagem SQL** (conceitos apresentados na Aula 07).
 - Usando os métodos de **pyspark.sql** (conceitos apresentados na Aula 08).
- Na listagem das respostas:
 - As **colunas** solicitadas devem ser exibidas exatamente na mesma ordem que a definida.
 - As **linhas** retornadas como respostas devem ser exibidas exatamente na mesma ordem que a definida.
 - Os **nomes das colunas** renomeadas devem seguir estritamente os nomes definidos.

▼ 3.1.2 Consulta OLAP (RESOLVER)

Liste, para cada código da equipe, setor do cliente e nome do estado do cliente, a soma das receitas. Arredonde a soma das receitas para até duas casas decimais. Devem ser exibidas as colunas na ordem e com os nomes especificados a seguir: "CODIGOEQUIPE", "SETORCLIENTE", "NOMEESTADOCLIENTE", "TOTALRECEITA". Ordene as linhas exibidas primeiro pelo total de receita em ordem ascendente, depois pelo código da equipe em ordem ascendente, depois pelo setor do cliente em ordem ascendente, depois pelo nome do estado em ordem ascendente.

```
# Escreva aqui a sua resposta para a consulta OLAP
```

```
# Caso tenha feito a sua resposta usando a linguagem SQL
# querySQL = "" comando SQL ""
# Transforme o resultado da consulta, o qual encontra-se em "querySQL",
# em um DataFrame em Pandas descomentando o comando a seguir:
```

```
# df = spark.sql(querySQL).toPandas()

# Caso tenha feito a sua resposta usando os métodos de pyspark.sql
# dataframepyspark = resposta em pyspark sem finalizar com o método .show()
# Transforme o resultado da consulta, o qual encontra-se
# em "dataframepyspark", em um dataframe em Pandas
# descomentando o comando a seguir:

# df = dataframepyspark.toPandas()

# Em qualquer caso, exiba algumas linhas do dataframe de nome df,
# o qual é um dataframe em Pandas, descomentando o comando a seguir:

# df
```

Note que o dataframe em Pandas, chamado `df`, é o dataframe a ser utilizado para o treinamento da rede neural.

▼ 3.2 Treinamento da Rede Neural

A segunda parte para solucionar a investigação consiste no treinamento da rede neural usando como base os dados históricos obtidos no item 3.1.2, segundo as instruções detalhadas a seguir.

▼ 3.2.1 Preparação dos Dados

A preparação dos dados já encontra-se pronta, sendo necessário apenas executar as células.

```
# ordena dados
df = df.sort_values(by='CODIGOEQUIPE')

# obtem dummy-variables / one-hot-encoding
setor = pd.get_dummies(df['SETORCLIENTE'])
estado = pd.get_dummies(df['NOMEESTADOCLIENTE'])
equipe = pd.get_dummies(df['CODIGOEQUIPE'])

# cria dataframe para treinamento
input_data = pd.concat([equipe, setor, estado, df['TOTALRECEITA']], axis=1, sort=False)

# separa dados de treinamento e 1 para teste
n_test = 1

x_train = np.array(input_data.iloc[:-n_test, :-1])
x_test = np.array(input_data.iloc[-n_test:, :-1])
```

```

y_train = np.array(input_data.iloc[:-n_test,-1])
y_test = np.array(input_data.iloc[-n_test,-1])

print("Tamanho conjunto de treinamento: ", x_train.shape)
print("Tamanho conjunto de testes: ", x_test.shape)

# valor maximo para normalizar receita
y_max = np.max(y_train)
print('Max train', y_max)

# dados para regressao normalizados
y_train = np.array(input_data.iloc[:-n_test,-1])/y_max
y_test = np.array(input_data.iloc[-n_test:,-1])/y_max

```

▼ 3.2.2 Projeto e Treinamento de Rede Neural (RESOLVER)

Utilize o código a seguir e, em seguida, compile e treine a rede neural utilizando:

- função de custo de erro médio quadrático,
- learning rate inicial de 0.005 e decaimento dado pela função `scheduler` provida,
- 30 épocas,
- `batchsize` de tamanho 10.

```

def deep_net(neurons, input_dim, n_layers=1, batch_norm=False, dropout_rate=0.0):

    input_data = keras.layers.Input(shape=(input_dim,))

    x = keras.layers.Activation('relu')(input_data)
    if batch_norm:
        x = keras.layers.BatchNormalization(name='bn_input')(x)
        x = keras.layers.Activation('relu')(x)
    x = keras.layers.Dense(neurons, activation='relu')(x)
    x = keras.layers.Dense(neurons, activation='relu')(x)
    x = keras.layers.Dropout(dropout_rate)(x)
    output = keras.layers.Dense(1, activation='relu')(x)
    dnn = keras.models.Model(input_data, output)

    return dnn

def scheduler(epoch, lr):
    if epoch < 5: return lr
    return lr * tf.math.exp(-0.05)

callbacklr = tf.keras.callbacks.LearningRateScheduler(scheduler)

seed(1)
set_seed(2)
dnn = deep_net(64, x_train.shape[1], batch_norm=False, dropout_rate=0.5)

```



```
# Escreva aqui a sua resposta para a compilação e treinamento de rede neural  
  
# compilar  
# treinar
```

▼ 3.2.3 Predição da receita (RESOLVER)

Utilizando o modelo treinado, realize a predição da receita para a linha separada para teste, i.e. a linha em `x_test`

Não se esqueça de, após obter a predição, reescalar novamente para o intervalo original dos dados, revertendo a normalização feita na etapa de preparação dos dados.

```
# Escreva aqui a sua resposta para a predição de receita em milhões de reais
```

