PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
NÚCLEO DE EDUCAÇÃO A DISTÂNCIA
Pós-graduação *Lato Sensu* em Ciência de Dados e Big Data

Cláudio Vasconcelos Braga

PREVISÃO DA DEMANDA DE CONSULTAS SQL AO AMBIENTE ANALÍTICO COM O OBJETIVO DE SUBSIDIAR CONTRATAÇÃO EM NUVEM

Belo Horizonte
2021

Cláudio Vasconcelos Braga

PREVISÃO DA DEMANDA DE CONSULTAS SQL AO AMBIENTE ANALÍTICO COM O OBJETIVO DE SUBSIDIAR CONTRATAÇÃO EM NUVEM

Trabalho de Conclusão de Curso apresentado ao Curso de Especialização em Ciência de Dados e Big Data como requisito parcial à obtenção do título de especialista.

Belo Horizonte
2021

SUMÁRIO

1. Introdução

1.1. Contextualização

Uma pessoa jurídica (PJ) de direito público possui um ambiente analítico que é utilizado por centenas de cientistas de dados. O nome desta PJ foi omitido com o objetivo de ajudar na preservação do sigilo das informações. Este ambiente é capaz de executar comandos SQL em grandes volumes de dados e foi especificado e dimensionado de forma a suportar todas as requisições feitas pelos usuários..

O Serpro é uma empresa pública prestadora de serviços de tecnologia da informação e atende diversos órgãos públicos. Toda a infraestrutura tecnológica e a operacionalização do ambiente analítico são contratados com o Serpro por um preço fixo mensal.

A figura 1 demonstra a utilização deste ambiente durante um período de 5 semanas.

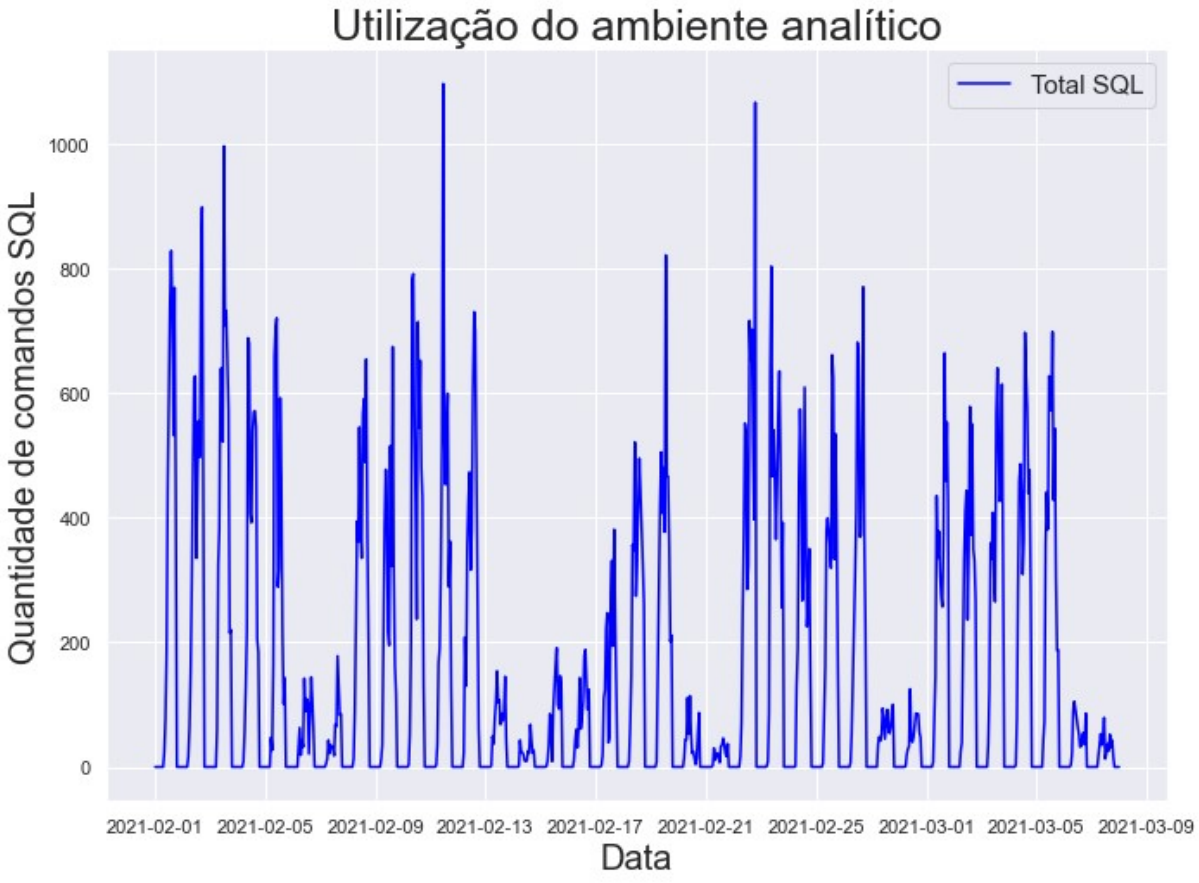

Figura 1 – Utilização do Ambiente Analítico

Pode-se facilmente observar uma variação significativa da quantidade de comandos SQL executada. Cada pico representa 1 dia. O maior volume ocorre nos dias de semana no período de expediente enquanto que o menor volume ocorre no sábado e no domingo.

Diversos provedores de serviço em nuvem oferecem Plataformas como Serviço (PaaS) capazes de processar grandes volumes de dados. Uma das maiores vantagens da contratação em nuvem é a capacidade de alterar a capacidade de processamento conforme a necessidade.

Com o objetivo principal de reduzir os custos com o ambiente analítico e melhorar a quantidade de serviços ofertados aos usuários, está em estudo a contratação de um ambiente analítico na nuvem que possa ser escalável de maneira rápida, possibilitando que, nos momentos de pouca demanda, haja uma menor capacidade de processamento alocada e conseqüentemente um menor custo.

1.2. O problema proposto

Este trabalho propõe obter a previsão do comportamento da série temporal das quantidades de consultas SQL feitas no ambiente analítico com o objetivo de subsidiar futura contratação PaaS em nuvem.

Este problema é importante, pois possibilitará o conhecimento do comportamento atual e futuro da demanda por consultas analíticas permitindo a escolha adequada de um serviço em nuvem e seu dimensionamento para suportar as oscilações da demanda prevista ao longo do tempo com o menor custo possível.

Os dados foram obtidos de duas fontes:

- Base de dados utilizada para monitoramento da execução das consultas dos usuários ao Impala (banco de dados de pouca latência e alta concorrência para o Haddop), chamada de CONSULTAS.
- Base de dados dos funcionários, chamada de USUARIOS.

A utilização da base USUARIOS possibilita o conhecimento mais aprofundado sobre o usuário que executou as consultas. Verificou-se que uma parcela destas consultas foi feita por funcionários do Serpro. Estas consultas devem ser

descartadas uma vez que o ambiente analítico em nuvem não deverá ser utilizado pelo Serpro.

A base de dados fonte dos dados da tabela CONSULTAS apresentava dados incorretos até o final de janeiro de 2021, data em que o problema foi identificado e corrigido. Os dados deste estudo abarcarão o período após a correção do problema, entre 1º de fevereiro e 9 de março de 2021.

A linguagem Python foi escolhida para o desenvolvimento deste trabalho pela variedade de bibliotecas existentes, possibilitando o aproveitamento das diversas funcionalidades disponíveis e reduzindo significativamente a necessidade de desenvolvimento de novas funcionalidades. Esta linguagem foi criada por Guido van Rossum, no Instituto Nacional de Pesquisa para Matemática e Ciência da Computação da Holanda (CWI), em 1991, sendo considerada uma linguagem de script orientada a objetos e interpretada.

O código abaixo inicializa todas as bibliotecas utilizadas neste trabalho.

```
from pandas import DataFrame
import pandas as pd
import numpy as np
import math

from itertools import product

from datetime import datetime
from datetime import timedelta

import chart_studio.plotly as py
import plotly.express as px
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

```
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout
from keras.callbacks import EarlyStopping
from keras.layers import *

from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import train_test_split

import statsmodels.api as sm
from statsmodels.graphics.tsaplots import plot_pacf
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.tsa.holtwinters import ExponentialSmoothing
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.seasonal import seasonal_decompose

from tqdm import tqdm_notebook

import warnings
warnings.filterwarnings('ignore')
```

2. Coleta de Dados

Como mencionado anteriormente este trabalho utiliza como fonte de dados uma tabela sobre as consultas executadas chamada de CONSULTAS e uma tabela sobre os funcionários, chamada de USUARIOS.

O campo USER da tabela CONSULTAS e o campo CPF da tabela USUÁRIOS possibilitam a junção entre as duas tabelas. Importante ressaltar que cada usuário pode executar 0 ou mais consultas SQL e que podem existir consultas com usuários que não estão na tabela USUÁRIOS. A figura 2 apresenta o diagrama de entidades e relacionamento destas tabelas.

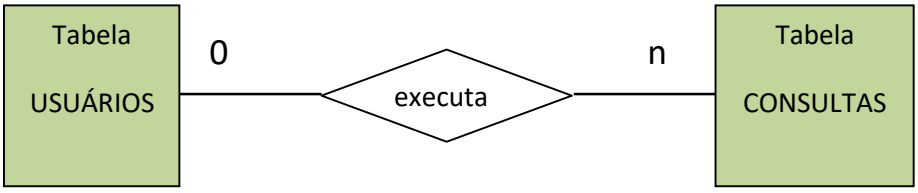

Figura 2 – Diagrama de Entidades e Relacionamento

2.1. Tabela CONSULTAS

Os dados referentes aos comandos SQL foram coletados de um banco de dados Infobright e exportados para um arquivo texto com os campos separados por virgula (CSV). Segue abaixo o comando utilizado neste processo.

```
SELECT query_id,
           queryState ,
           user,
           data_hora
FROM     impala_queries
WHERE   queryType = ’QUERY’ AND
           data_hora  > '2021-02-01 00:00:00’
```

Foram recuperados 161957 registros. A descrição das informações dos campos e seus tipos são apresentados na tabela 1.

CAMPO	DESCRIÇÃO	TIPO
query_id	Identificação da solicitação de execução SQL no Impala.	Texto
queryState	Estado da execução da consulta, pode assumir os seguintes valores: Finished, Exception, Lost, Compiled e Running.	Texto
data_hora	Data e hora do recebimento do comando SQL	Texto
User	Usuário que executou o comando SQL. Na maior parte dos casos é formado pelo CPF, mas pode também ser um texto alfanumérico ou possuir um sufixo @DATALAKE.SERPRO após o CPF. Este campo foi criptografado.	Texto

Tabela 1 – Detalhamento dos campos da tabela CONSULTAS

2.2. Tabela USUÁRIOS

Os dados referentes aos usuários foram coletados de uma réplica do Data Warehouse existente no ambiente analítico e exportados para um arquivo texto com os campos separados por virgula (CSV). Segue o comando SQL utilizado neste processo:

```
SELECT  cd_cpf_rec_humano cpf,
           Tp_rec_humano orgao
FROM     wd_rec_humano
```

Foram recuperados 80300 registros. A descrição das informações dos campos e seus tipos são apresentados na tabela 2.

CAMPO	DESCRIÇÃO	TIPO
CPF	CPF do funcionário, com 11 posições. Este campo foi criptografado.	Texto
Órgão	Campo texto com duas posições que pode assumir os seguintes valores: ES, RF, RQ, SP e TC.	Texto

Tabela 2 – Detalhamento dos campos da tabela USUARIOS

3. Processamento e Tratamento de Dados

Os dados coletados na etapa anterior foram carregados, analisados, ajustados e depois foram juntados. O detalhamento deste processo é descrito nas seções a seguir. Importante ressaltar que os CPF foram criptografados para proteger o sigilo dos servidores.

3.1. Carga dos dados

Primeiramente, os dados das tabelas CONSULTAS e USUÁRIOS foram carregados no Python:

```
#Carrega tabela CONSULTAS
consultas= pd.read_csv('D:\\tcc\\QUERIES_9MAR_AJ.csv')

#Carrega tabela USUÁRIOS
usuarios= pd.read_csv('D:\\tcc\\USUARIOS_9MAR_AJ.csv',dtype={'cpf':str,'orgao':str})
usuarios.rename(columns={'cpf':'cpf_usuario'}, inplace=True)
```

3.2. Estrutura das tabelas e ausência de dados.

Os comandos a seguir exibem as estruturas das tabelas CONSULTAS e USUARIOS, confirmando a carga de todos os registros existentes nos arquivos CSV.

```
consultas.info()
usuários.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 161957 entries, 0 to 161956
Data columns (total 4 columns):
 #   Column      Non-Null Count   Dtype
---  ------      --------------   -----
 0   query_id    161957 non-null  object
 1   queryState  161957 non-null  object
 2   user        161957 non-null  object
 3   data_hora   161957 non-null  object
dtypes: object(4)
memory usage: 4.9+ MB

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 80300 entries, 0 to 80299
Data columns (total 2 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   cpf_usuario  80300 non-null  object
 1   orgao        80299 non-null  object
dtypes: object(2)
memory usage: 1.2+ MB
```

Em relação a tabela USUARIOS, pode-se observar que o campo ORGAO apresenta uma ocorrência de ausência de valores. O comando abaixo identifica este registro e exibe o seu conteúdo.

```
ausencia= usuarios.apply(lambda row: row.orgao!=row.orgao, axis=1)
usuarios[ausencia]
```

Reg	cpf_usuario	orgao
76884	Não se aplica	NaN

Tabela 3 – Resultado da consulta

Optou-se por excluir este registro.

```
usuarios.drop(index=76884,inplace=True)
```

3.3. Ajuste do campo USER da tabela CONSULTAS

O campo USER algumas vezes apresenta, após o CPF do usuário, o texto “@DATALAKE.SERPRO”. Para solucionar esta inconsistência foi criado o campo CPF_USUARIO com o conteúdo do campo USER excluído este texto adicional quando ele ocorre.

```
def corrige_usuario(row):
   resp=row.user
   if '@DATALAKE.SERPRO' in row.user:
      resp=row.user[:-16]
   return(resp)

consultas['cpf_usuario']=consultas.apply(corrige_usuario,axis=1)
```

3.4. Inclusão do usuário “Contagil” na tabela USUARIOS.

Nem todas as consultas são executadas por usuários, uma parcela delas é executada por um sistema chamado ContÁgil, utilizando a conta de sistema chamada “contagil”.

A tabela USUARIOS não possui este usuário, contudo, para efeito deste estudo, precisamos incluir este registro para que haja uma correspondência entre as tabelas quando os dados forem juntados.

```
usuarios=usuarios.append({'cpf_usuario':'contagil','orgao':'RF'},ignore_index=True)
```

3.5. Exclusão das consultas não finalizadas

Analisando a distribuição do campo QUERYSTATE, constatamos que 92% das consultas foram concluídas, e que 6% apresentaram erro. A figura 3 apresenta esta distribuição.

```
QueryState_qtd= consultas['queryState'].value_counts()
plot=QueryState_qtd.plot.bar(title="Distribuição de QueryState")
print (QueryState_qtd)
```

```
FINISHED     149851
EXCEPTION      9434
LOST           2416
COMPILED        249
RUNNING           7
Name: queryState, dtype: int64
```

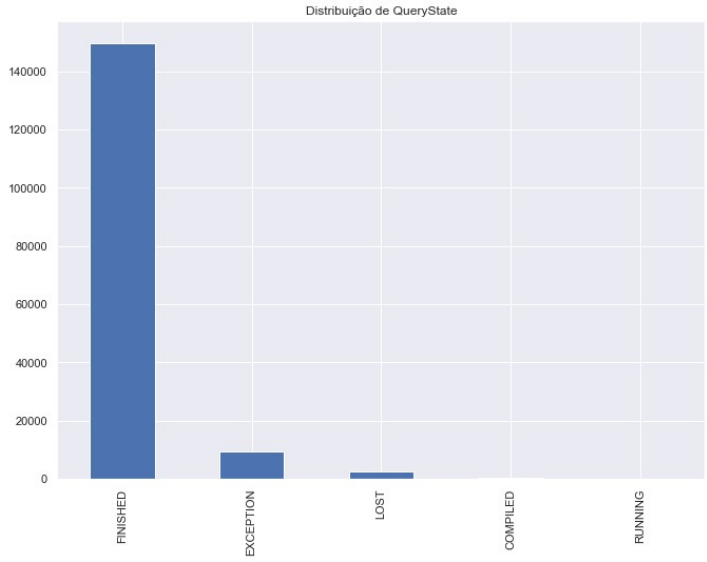

Figura 3 – Distribuição do campo QUERYSTATE

As consultas nas situações “EXCEPTION” representam consultas que apresentam erro de sintaxe ou que a quantidade de recursos computacionais do ambiente analítico não foi capaz de executar o comando solicitado. Em ambos os casos, o usuário costuma refazer a consulta. Desta forma, optou-se por excluir todos os registros com esta situação.

As consultas nas situações “LOST” representam consultas que, por algum motivo desconhecido, não foi possível o acompanhamento das suas execuções, sabe-se apenas que o usuário não recebeu o resultado. Estas situações também foram excluídas.

```
consultas=consultas.loc[(consultas.queryState !="EXCEPTION") & (consultas.queryState !="LOST"
),['cpf_usuario','query_id','data_hora']].copy().reset_index(drop=True)
```

3.5. Correção do fuso horário do campo DATA_HORA

Com objetivo de testar a integridade dos dados da tabela CONSULTAS foram executadas algumas consultas no ambiente analítico e depois conferido o conteúdo da tabela.

Nestes testes foi constatado que o campo DATA_HORA está em um fuso horário diferente, resultando em uma hora a mais em relação ao fuso horário de Brasília-DF.

O código abaixo cria o campo INICIO a partir da conversão do tipo do campo DATA_HORA e faz o ajuste no fuso horário. O campo DATA_HORA foi descartado.

```
# conversão do tipo do campo Data_hora de String para o Time
def str2time(val):
   try:
      return datetime.strptime(val,'%Y-%m-%d %H:%M:%S')
   except:
      return pd.NaT
consultas['inicio']= consultas.apply(lambda x: str2time(x.data_hora),axis=1)

# Redução de 1 hora
consultas.loc[:,'inicio'] = consultas.inicio - timedelta(minutes=60)

# exclusão do campo data_hora
consultas.drop(columns=['data_hora'],inplace=True)
```

3.6. Junção das tabelas CONSULTAS e USUÁRIOS

Os campos das tabelas CONSULTAS e USUARIOS estão apresentados na tabela 4. Pode-se observar que o campo CPF_USUARIO é comum nas duas tabelas e serve como elemento de ligação entre elas.

TABELA CONSULTAS	
CAMPO	**DESCRIÇÃO**
cpf_usuario	CPF do usuário, chave de ligação
inicio	Horário de inicio da consulta
query_id	Identificação da consulta

TABELA USUARIOS	
CAMPO	**DESCRIÇÃO**
cpf_usuario	CPF do usuário, chave de ligação
Orgao	Órgão do funcionário.

Tabela 4 – Campos das tabelas CONSULTAS e USUARIOS

A junção será realizada utilizando o LEFT JOIN, ou seja, preserva-se todos os registros da tabela CONSULTAS e quando não houver correspondência na tabela USUARIO, preenche o conteúdo com NAN.

```
# Left join.
juncao=pd.merge(consultas,usuarios,how="left")

# Seleciona registros onde orgao é NAN, ou seja, não CPF não foi encontrado na tabela usuarios
juncao['desconhecido']= juncao.apply(lambda row: row.orgao!=row.orgao, axis=1)

# Mostra relação de CPF que não foram encontrados na tabela usuarios
juncao.loc[juncao.desconhecido,'cpf_usuario'].unique()
```

A relação de CPF que não foram encontrados na tabela usuarios não foi apresentada para proteger o sigilo destes servidores. Cabe informar que os CPF listados são de funcionários do SERPRO ou representam contas de serviço de processos desenvolvidos pelo SERPRO.

Foram encontrados 14 CPF e 6 contas de serviço nesta situação. Nestes casos, o campo ORGAO foi definido como “perstadorSV”. O código a seguir realiza estes procedimentos e apresenta a distribuição final do campo ORGAO.

```
# Atribui "prestadorSV" a orgão nos casos onde que não houve correspondência em USUARIOS
juncao.loc[juncao.desconhecido,'orgao']="prestadorSV"
juncao.drop(columns=['desconhecido'],inplace=True)

# Apresenta a distribuição final do campo ORGAO
juncao.orgao.value_counts()
```

```
RF             133232
prestadorSV     16868
TC                  7
Name: orgao, dtype: int64
```

Primeiramente, cabe esclarecer o significado dos elementos do domínio do campo ORGAO.

- RF: Servidores que pertencem ao quadro de funcioários, ou o serviço ContÁgil incluido no item 3.4 deste trabalho.

- TC: Servidores de outro órgão da admininstração pública.
- PrestadorSV: Funcionários do Serpro ou serviços executados pelo Serpro.

As consultas geradas por funcionários do Serpro ou serviços executados pelo Serpro não serão considerados neste trabalho pois estas consultas não deverão ser executadas no novo ambiente analiítico a ser contratado. Portando serão excluídos todos os registros com ORGAO igual a “prestadorSV”.

```
dados=juncao.loc[(juncao.orgao!="prestadorSV"),['query_id','inicio']].reset_index(drop=True)
```

3.7. Agrupamento dos dados por período de tempo definido

Uma nova base de dados foi gerada com o quantitativo de consultas a cada 60 minutos. Além disso, foram definidas as variáveis periodo_dia e periodo_semana que indicam respectivamente a quantidade de registros em um dia (24) e em uma semana (168).

```
#DEFINIÇÃO DE PERÍODO
qtd_partes_1hora=1
delta=timedelta(minutes=60/qtd_partes_1hora)
periodo_dia=24*qtd_partes_1hora
periodo_semana=7*periodo_dia

#DEFINICAO DO INICIO DO INTERVALO DE ANÁLISE
inicioAnalise=datetime.strptime('2021-02-01 00:00:00','%Y-%m-%d %H:%M:%S')

#DEFINICAO DO FIM DO INTERVALO DE ANÁLISE
fimAnalise=str(df2.sort_values('inicio').inicio.tail(1).values[0])
fimAnalise=fimAnalise[0:10]+" "+fimAnalise[11:19]
fimAnalise=datetime.strptime(fimAnalise,'%Y-%m-%d %H:%M:%S')
```

```
#Gera nova base
dados=pd.DataFrame( {'Tempo':[],'qtd':[]})

momento=inicioAnalise
while momento<fimAnalise:
   qtd=dados_pre[(dados_pre.inicio >= (momento)) &
         (dados_pre.inicio <  (momento+delta))]['query_id'].count()
   dados = dados.append({'Tempo':momento,'qtd':qtd}, ignore_index=True)
   momento+=delta
```

4. Análise e Exploração dos Dados

Séries temporais são coleções de dados ordenados no tempo (Box et al. 2013). A tabela DADOS armazena 882 registros e pode ser caracterizada como uma série temporal pois possui 2 campos: TEMPO e QTD.

```
dados.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 882 entries, 0 to 881
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Tempo   882 non-null    datetime64[ns]
 1   qtd     882 non-null    float64
dtypes: datetime64[ns](1), float64(1)
memory usage: 13.9 KB
```

A figura 4 apresenta o conteúdo da série graficamente.

```
# Mostra a utilização do ambiente analítico
sns.set(rc={'figure.figsize':(11, 8)})
plt.plot(dados.loc[:,'Tempo'],dados.loc[:,'qtd'], color = 'blue', label = 'Total SQL')
plt.title('Utilização do ambiente analítico', fontsize=25)
plt.xlabel('Data', fontsize=20)
plt.ylabel('Quantidade de comandos SQL', fontsize=20)
params = {'legend.fontsize': 15}
plt.rcParams.update(params)
plt.legend()
plt.show()
```

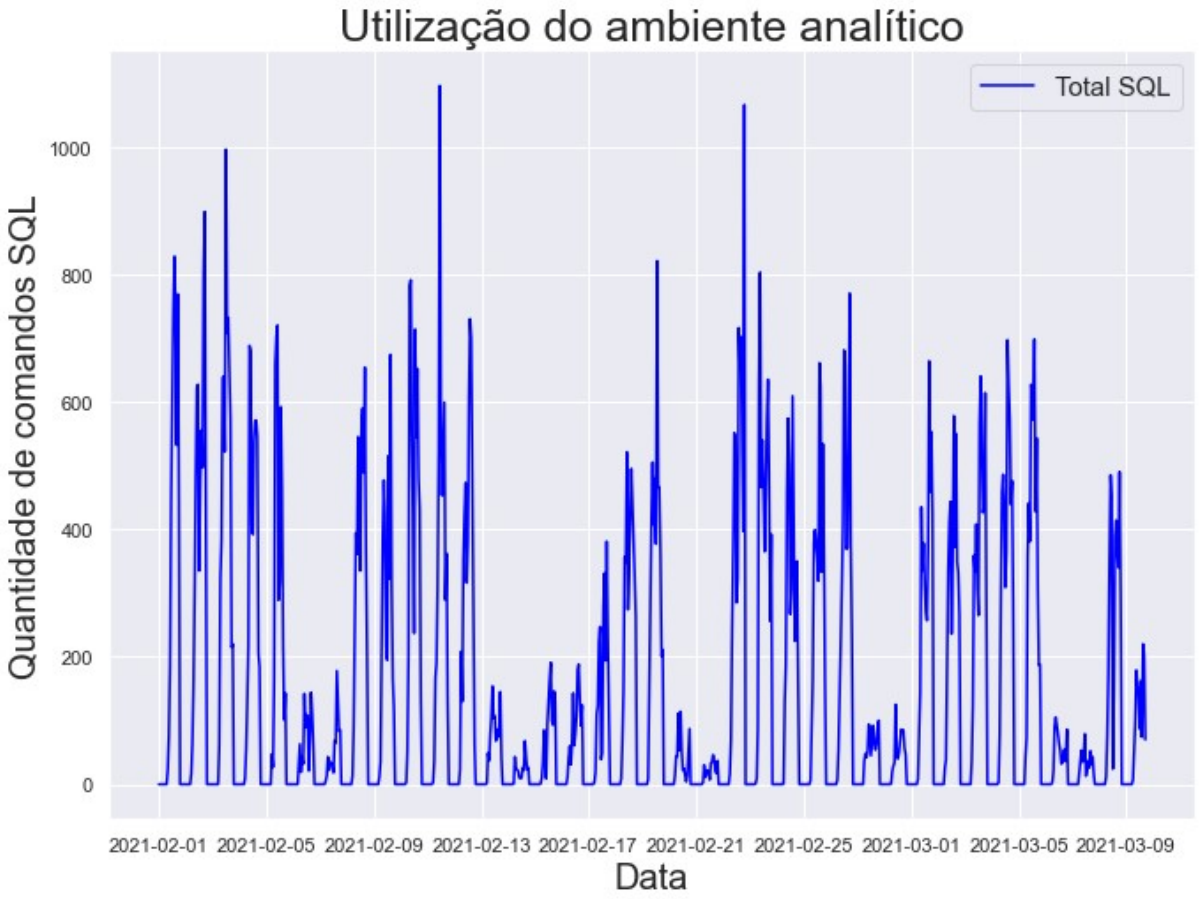

Figura 4 – Comportamento da série DADOS.

Podemos observar na figura 4 que cada um dos picos no gráfico representa um dia e que existe um padrão que se repete a cada 7 dias. Dividindo-se o numero de registros (882) por 24, podemos inferir que a base de dados armazena 36 dias e algumas horas.

```
sns.set(rc={'figure.figsize':(5, 4)})
fig, axs = plt.subplots(qtd_periodo)
fig.suptitle('Gráfico Semanal')

for loop in range(qtd_periodo):
   axs[loop].plot(dados.qtd[periodo*loop:periodo*(loop+1)-1])
   axs[loop].grid(False)
   axs[loop].set_yticks([])
   axs[loop].set_xticks([])
```

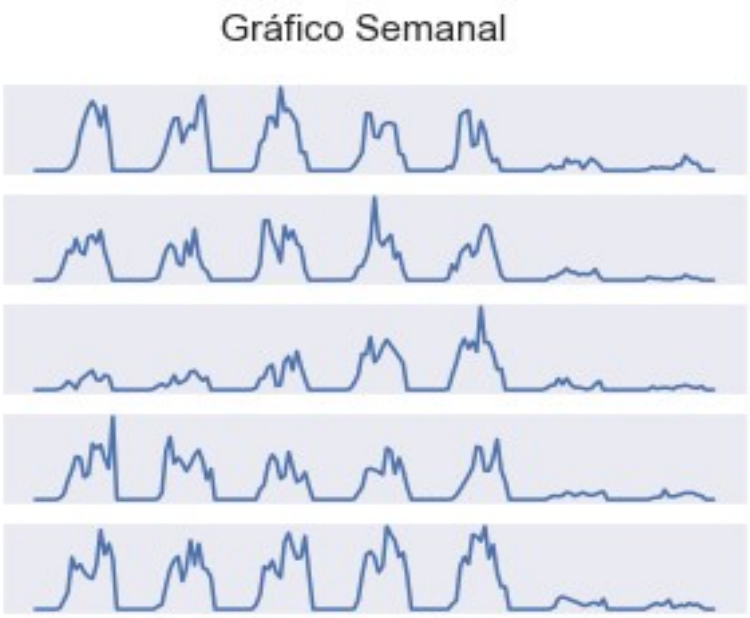

Figura 5 – Comportamento da semanal.

Na figura 5, é possível observar um comportamento atípico da terceira semana, que corresponde a semana do carnaval de 2021. Nesta semana o volume de consultas SQL somente começou a se comportar de maneira similar as demais semanas após a quarta-feira de cinzas.

Os dados da terceira semana foram descartados para evitar que a previsão do comportamento futuro da série temporal considere o carnaval. A figura 6 apresenta o comportamento da série sem o carnaval.

```
# Exclui o período do carnaval
antes=dados[:periodo*2-1]
depois=dados[periodo*3:qtd_periodo*periodo]
dados=antes.append(depois).reset_index(drop=True)
qtd_periodo-=1
# Exibe os novos dados
sns.set(rc={'figure.figsize':(11, 8)})
plt.plot(dados.loc[:,'qtd'].values, color = 'blue', label = 'Total SQL')
plt.title('Utilização do ambiente analítico sem Carnaval', fontsize=25)
plt.xlabel('Data', fontsize=20)
#plt.xticks(rotation=90)
plt.ylabel('Quantidade de comandos SQL', fontsize=20)
params = {'legend.fontsize': 15}
plt.rcParams.update(params)
plt.legend()
plt.show()
```

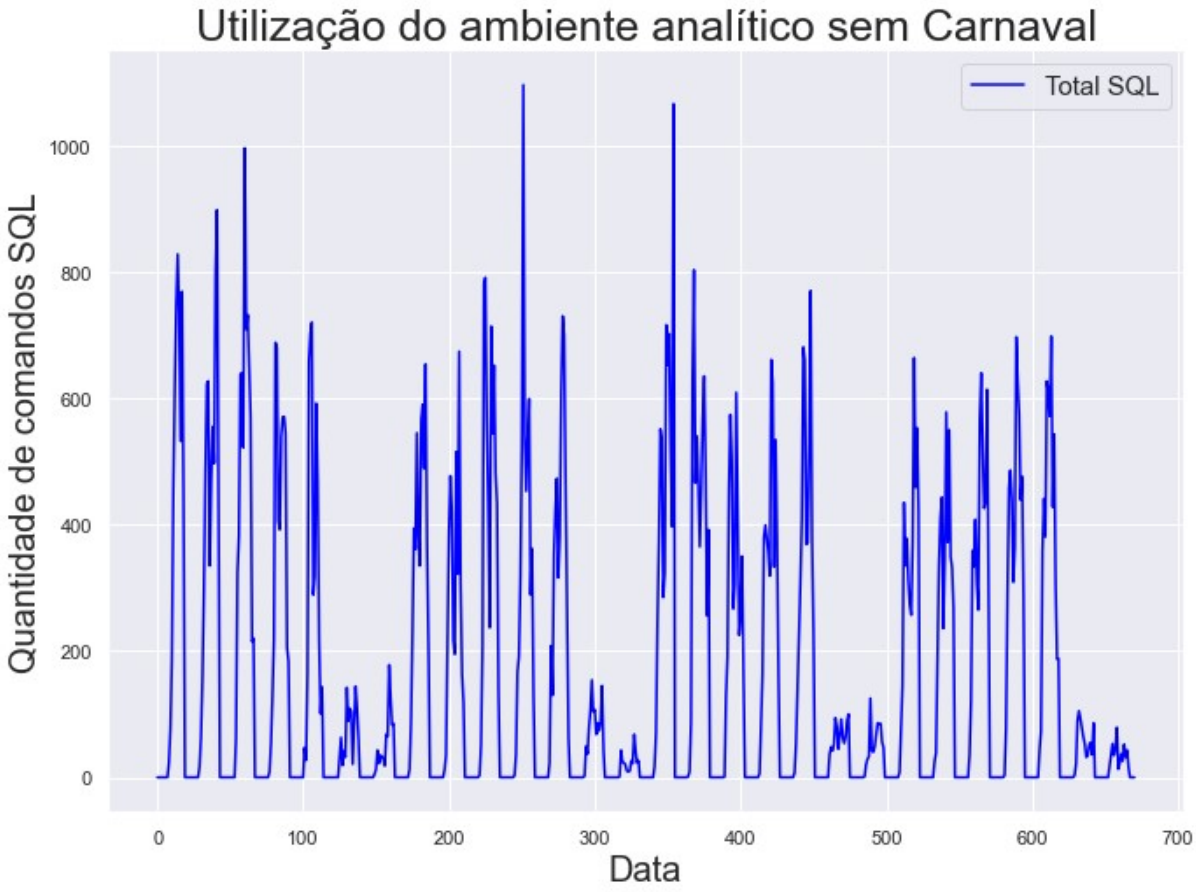

Figura 6 – Comportamento da série sem o carnaval.

5. Criação de Modelos de Machine Learning

Este trabalho propõe obter a previsão do comportamento da série temporal das quantidades de consultas SQL feitas no ambiente analítico com o objetivo de subsidiar futura contratação PaaS em nuvem. Ou seja, utilizar os dados que já temos para prever os valores futuros.

Primeiramente a série temporal foi dividida em duas partes: (i) treinamento e (ii) teste. A primeira parte tem como objetivo ser utilizada para treinar o modelo enquanto que a segunda parte será utilizada para medir a eficiência do modelo.

```
#Segmenta os dados
total=dados.loc[:ultimo_periodo-1,'qtd'] # 3 semanas
treinamento = dados.loc[:((qtd_periodo-1)*periodo-1), 'qtd'] # 2 semanas
teste= dados.loc[(qtd_periodo-1)*periodo:(qtd_periodo)*periodo-1, 'qtd'] # 1 semana

#Exibe os dados
plt.plot(treinamento, color = 'red', label = 'treinamento')
plt.plot(teste, color = 'blue', label = 'teste')
plt.title('Segmentação da base de dados',fontsize=20)
plt.xlabel('Data',fontsize=15)
plt.ylabel('Tamanho na fila',fontsize=15)
params = {'legend.fontsize': 15}
plt.rcParams.update(params)
plt.legend()
plt.show()
```

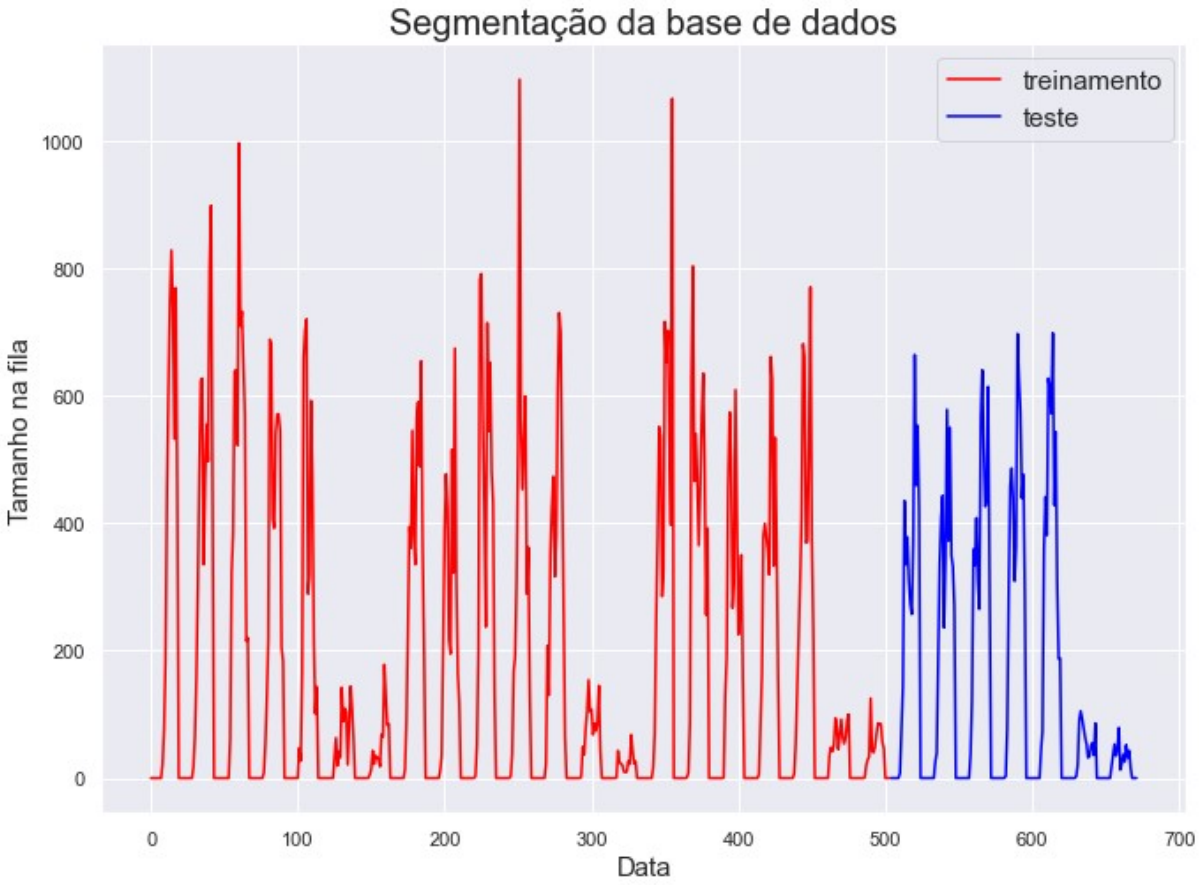

Figura 7 – Segmentação da base de dados

Foram criados dois modelos de aprendizado de máquina objetivando prever o comportamento futuro da série temporal DADOS. No primeiro modelo foi utilizada a técnica SARIMA enquanto que no segundo modelo foi utilizado a técnica LSTM.

5.1. Modelo SARIMA

Uma série temporal pode ser dividida em três componentes: (i) tendência, (ii) sazonalidade e (iii) resíduo. A tendência retrata o comportamento de longo prazo de uma série temporal. A sazonalidade é uma oscilação que se repete regularmente dentro de um período de tempo específico. Por fim, o resíduo é a parcela que não é explicada pelos outros dois componentes da série temporal (Weigend, 1994).

```
#Decomposição da Série temporal
result = seasonal_decompose(treinamento.values,period=periodo_semana)
fig = result.plot()
```

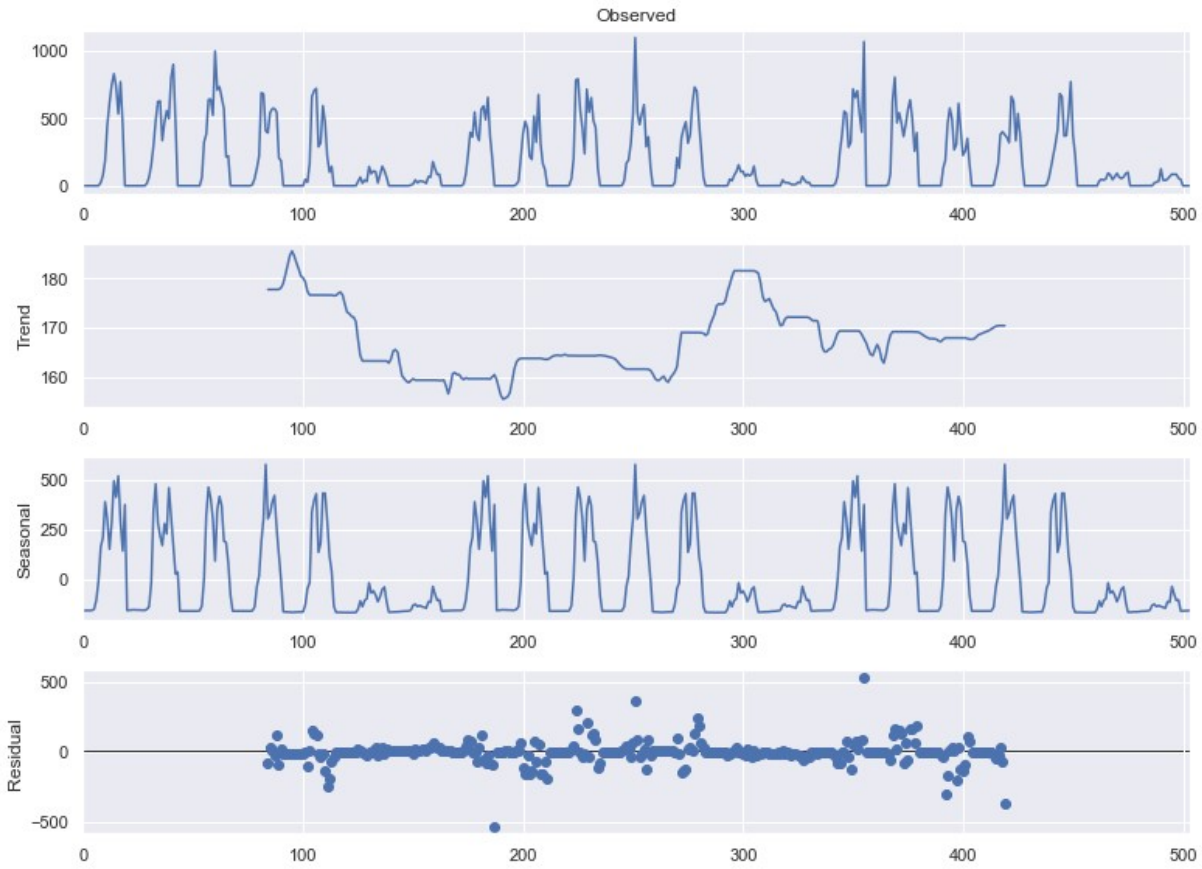

Figura 8 – Decomposição da Série Temporal

O modelo autoregressivo integrado de médias móveis (ARIMA) é muito utilizado para análises de séries temporais (Lee; Koo, 2011). Um dos pressupostos deste modelo é a necessidade dos dados serem estacionários, ou seja, a média,

variância e estrutura de autocorrelação não mudam no decorrer do tempo (Chatfield, 2002).

Segundo Box-Jenkins (1978), o modelo ARIMA pode ser subdivido em três partes:

- AR: Representa a parte auto-regressiva, é utilizado para ajustar o componente implícito de regressão linear.

- I: Representa a parte integrada, quanto maior, mais transformações de diferenciação são realizadas sobre os dados. Este componente é de suma importância para garantir a estacionáriedade do modelo.

- MA: Representa a ordem do componente de média móvel aplicado aos resíduos.

Para cada uma destas partes existe uma variável maior que zero associada. As variáveis p,d e q representam nesta ordem os três componentes.

Quando se acrescenta uma componente de sazonalidade, obtêm-se o modelo autoregressivo integrado de médias móveis sazonal (SARIMA), tendo coeficientes próprios associados a cada uma das partes (MAAROF et al., 2014). Pode ser representado como:

$$\text{SARIMA } (p,d,q) \times (P,D,Q)s, \quad (1)$$

Onde:

p = ordem autorregressiva da parte não sazonal;
d = número de diferenças não sazonais;
q = ordem de médias móveis da parte não sazonal;
P = ordem autorregressiva da parte sazonal;
D = número de diferenças sazonais;
Q = ordem de médias móveis da parte sazonal; ´
S = sazonalidade da série.

Segundo Caldwell (1971), uma série não estacionária pode ser normalizada e transformada em estacionária subdividindo-a em períodos e realizando a diferenciação dos períodos para descobrir uma tendência. Este é um processo fundamental na aplicação de modelos ARIMA.

A estacionariedade da série temporal analisada neste trabalho pode ser melhorada aplicando a diferença entre os elementos em períodos subseqüentes. Ou seja, subtrair a quantidade de consultas SQL ocorrida em um dia e horário específico com o valor obtido no mesmo horário 7 dias depois.

O teste ADF (Dickey-Fuller Aumentado) é um teste estatístico de hipótese nula. Quando o p-value é inferior ao intervalo de confiança (0,25% neste trabalho) rejeita-se a hipótese nula que diz que a série tem raiz unitária e não é estacionária (MODENESI, 2008).

O código abaixo elimina a estacionariedade através da subtração dos termos correspondentes de períodos subsequentes e exibe o gráfico da série ajustada e o resultado do teste ADF.

```
treinamento_aj = treinamento.diff(período)[periodo:].reset_index(drop=True)

# Calcula a estatistica ADF
ad_fuller_result = adfuller(treinamento_aj)
print('ADF Statistic:{} ,p-value:{}'.format(ad_fuller_result[0],ad_fuller_result[1]))

#Exibe o Gráfico da série ajustada
treinamento_aj.plot()
```

```
ADF Statistic:-12.534203448190095 ,p-value:2.3916245921394147e-23
```

Figura 9 – Série ajustada pela diferenciação de ordem 1

O teste ADF nos permite concluir que a hipótese nula deve ser rejeitada, ou seja, a série pode ser considerada estacionária. O gráfico da série ajustada corrabora este entendimento pois não apresenta tendências de alteração de média, variância e estrutura de autocorrelação ao longo do tempo.

Portanto, como a estacionaridade foi obtida com apenas uma diferenciação, podemos concluir que existe uma grande possibilidade do parâmetro d ser igual a 1.

Os gráficos ACF e PACF podem ajudar na determinação dos componentes AR e MA do modelo ARIMA. A figura 10 apresenta estes gráficos.

```
plot_pacf(treinamento_aj);
plot_acf(treinamento_aj);
```

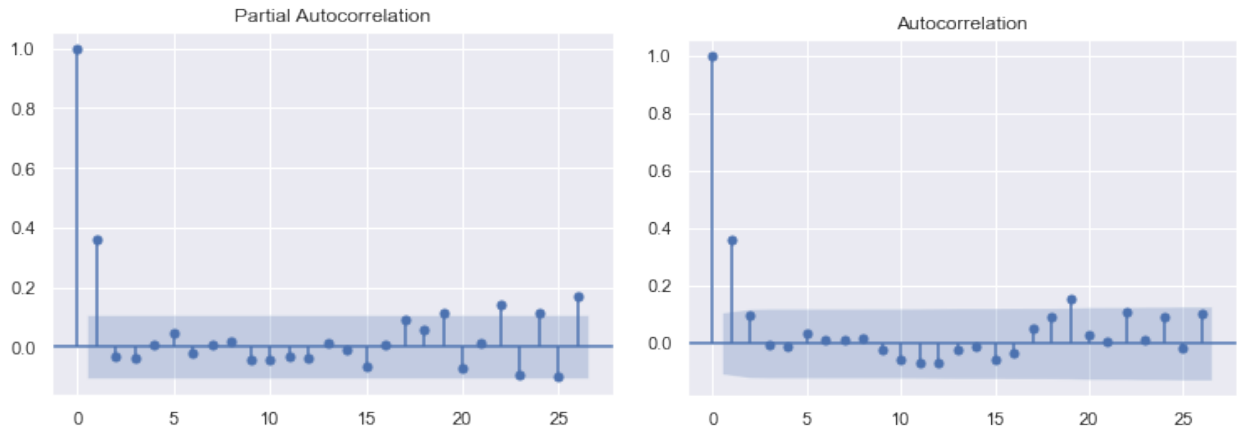

Figura 10 – Gráficos ACF e PACF

Poderíamos obter os restantes dos parâmetros utilizando os gráficos apresentados na figura 10, contudo para evitar erros de interpretação dos gráficos optou-se por uma abordagem de força bruta, ou seja, testar todas as possibilidades de parâmetros em um intervalo razoável.

Para isso foi definido a função grid_SARIMA que gera um modelo para cada um dos conjuntos de parâmetros passados e armazena em uma tabela um indicador da qualidade do modelo. O indicador escolhido foi o Akaike Information Criterion (AIC), quanto menor o seu valor melhor é o modelo.

```
def grid_SARIMA(parameters_list, periodo, exog):

    results = []
    for param in parameters_list:
        try:
            model = SARIMAX(exog, order=(param[0], param[1], param[2]), seasonal_order=(param[3],
                    param[4], param[5], periodo)).fit(disp=-1)
        except:
            continue
        aic = model.aic
        results.append([param, aic])
        print (param,aic)
    result_df = pd.DataFrame(results)
    result_df.columns = ['(p,d,q)x(P,D,Q)', 'AIC']

    #Ordena na ordem crescente
    result_df = result_df.sort_values(by='AIC', ascending=True).reset_index(drop=True)
    return result_df
```

A função grid_SARIMA recebe três informações: (i) lista de parâmetros, (ii) período e (iii) serie a ser treinada. A lista de parâmetros apresenta 256 diferentes combinações conforma intervalos definidos na tabela 5. O período indica a sazonalidade da série, ou seja, no caso concreto aqui tratado, o período é de 168, obtido pela multiplicação de 24 horas por 7 dias (uma semana).

Parâmetro	Valor inicial	Valor Final
p	0	4
d	0	2
q	0	4
P	0	4
D	0	2
Q	0	4

Tabela 5 – Lista de parâmetros testados

```
# Roda um grid com 256 possibilidades distinatas para procurar o SARIMA com melhor AIC
p = range (0,5,1)
d = range (0,2,1)
q = range (0,5,1)
P = range (0,5,1)
D = range (0,2,1)
Q = range (0,5,1)
parameters = product(p, d, q, P,D,Q)
parameters_list = list(parameters)

result_df11 = grid_SARIMA(parameters_list, periodo, treinamento.values)
result  df11.head(5)
```

(p,d,q)x(P,D,Q)	AIC
(2, 1, 2, 1, 1, 0)	4317.10
(2, 0, 2, 1, 0, 0)	4318.01
(2, 0, 2, 1, 1, 0)	4321.80
(0, 1, 0, 0, 1, 1)	4430.04
(0, 1, 0, 1, 1, 0)	4430.65

Tabela 6 – Melhores modelos

O melhor modelo é o que apresenta menor AIC. Portanto, o modelo escolhido foi o SARIMA $(2,1,2)(1,1,0)_{168}$.

```
model = SARIMAX(treinamento.values, order=(2, 1, 2), seasonal_order=(1, 1, 0, periodo)).fit()
print(model.summary())
```

```
                                SARIMAX Results
==========================================================================================
Dep. Variable:                                 y   No. Observations:                  504
Model:             SARIMAX(2, 1, 2)x(1, 1, [], 168)   Log Likelihood               -2152.550
Date:                              Thu, 11 Mar 2021   AIC                           4317.100
Time:                                      13:17:20   BIC                           4339.985
Sample:                                           0   HQIC                          4326.223
                                                - 504
Covariance Type:                                opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.6271      0.060    -10.455      0.000      -0.745      -0.510
ar.L2          0.3728      0.033     11.312      0.000       0.308       0.437
ma.L1         -0.0008      0.279     -0.003      0.998      -0.547       0.545
ma.L2         -0.9992      0.061    -16.389      0.000      -1.119      -0.880
ar.S.L168     -0.4157      0.038    -10.826      0.000      -0.491      -0.340
sigma2      2.014e+04   1.19e-05    1.7e+09      0.000    2.01e+04    2.01e+04
==========================================================================================
Ljung-Box (L1) (Q):                   0.17   Jarque-Bera (JB):              2017.49
Prob(Q):                              0.68   Prob(JB):                         0.00
Heteroskedasticity (H):               0.50   Skew:                             0.99
Prob(H) (two-sided):                  0.00   Kurtosis:                        14.86
==========================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
[2] Covariance matrix is singular or near-singular, with condition number 3.9e+24. Standard
errors may be unstable.
```

Por fim, o modelo é utilizado para prever o comportamento futuro da série. Os valores previstos são comparados com os valores de teste.

```
# Faz a previsão
forecast1 = model.predict()

# gera o gráfico comparando os resultados
sns.set(rc={'figure.figsize':(22, 8)})
plt.plot(forecast1[0:periodo], color = 'red', label = 'previsto SARIMA')
plt.plot(teste.values, color = 'green', label = 'Teste set')
plt.title('Comparativo predito e teste',fontsize=20)
plt.xlabel('Data',fontsize=15)
plt.ylabel('Quantidade de comandos',fontsize=15)
params = {'legend.fontsize': 15}
plt.rcParams.update(params)
plt.legend()
plt.show()
```

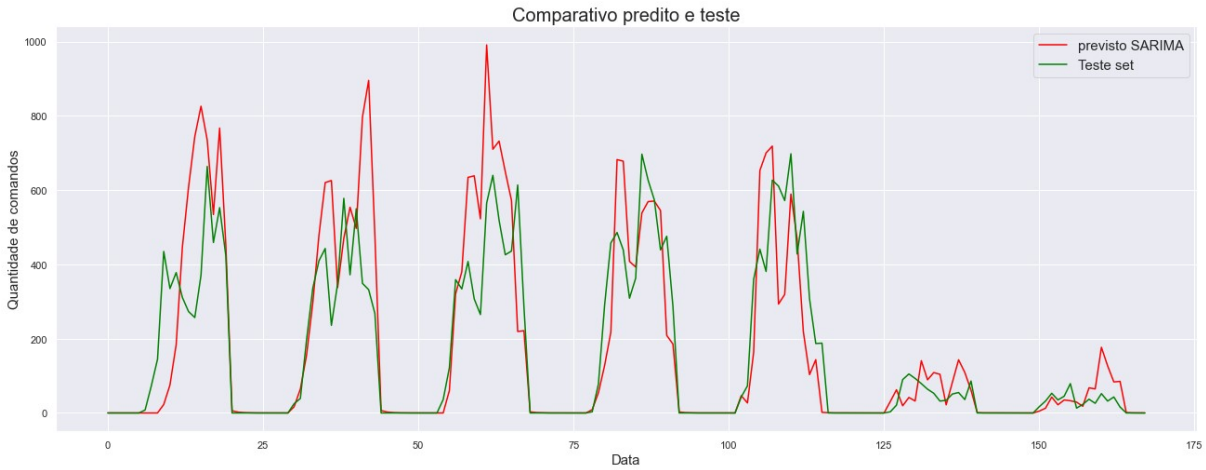

Figura 11 – Modelo SARIMA – teste VS predição

A figura 11 deixa claro que o modelo conseguiu realizar uma previsão adequada do comportamento futuro da série temporal. Com o objetivo de medir esta eficiência, foi calculada a raiz quadrada da média dos erros quadráticos das diferenças obtidas entre os valores reais e previstos (RMSE).

```
import math
# evaluate forecasts
rmse = math.sqrt(mean_squared_error(forecast1[0:periodo], teste.values))
print('Test RMSE: %.3f' % rmse)
```

```
Test RMSE: 141.938
```

5.2. Modelo LSTM

Segundo Havkin (2001), as Redes Neurais foram criadas baseadas no comportamento do cérebro humano. Enquanto que no cérebro existem milhões de neurônios conectados por sinapses, nas redes neurais são utilizados simuladores de neurônios, conectados de uma maneira similar aos neurônios do cérebro humano. A força destas conexões pode variar em resposta a estímulos, permitindo o aprendizado.

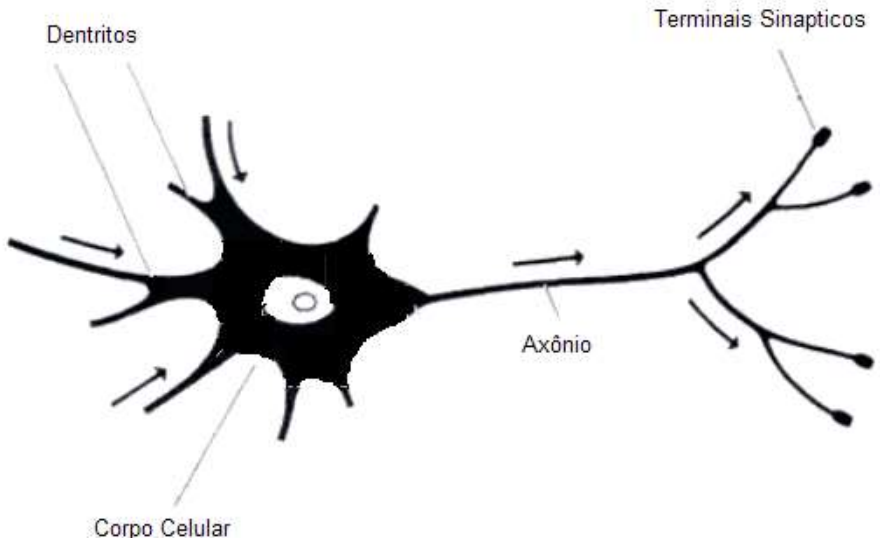

Figura 12 – Representação de um neurônio
Fonte: Adaptada de Arbib(2003)

Ainda segundo o autor, pode-se dizer que uma rede neural artificial se assemelha ao cérebro, pois o conhecimento é adquirido através de um processo de aprendizagem e são utilizadas forças de conexão entre os neurônios para armazenar o conhecimento adquirido. A figura 15 exemplifica um modelo não linear de um neurônio artificial.

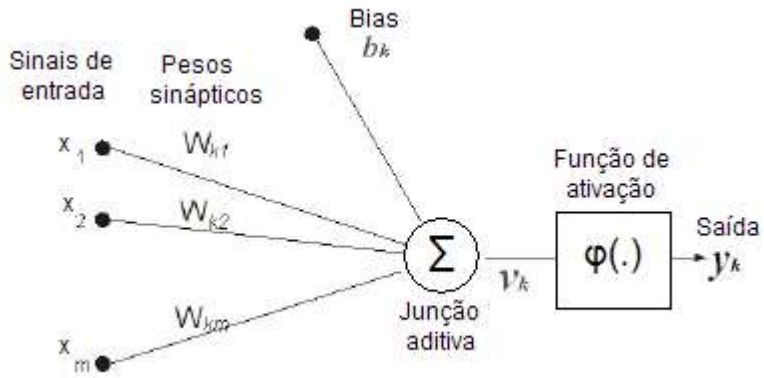

Figura 15 – Modelo não linear de um neurônio
Fonte: Haykin (2001)

Segundo Olah, C (2015), as diferentes formas de disposição e estruturação dos neurônios artificiais são chamadas de topologia da rede. Muitas redes, com diferentes topologias e formas de treinamento, foram desenvolvidas, ao longo do tempo, para se adaptarem aos diversos tipos de problemas a serem resolvidos.

Dentre elas destacam-se as redes neurais com células Long-Short Term Memory (LSTM). Estas redes são retroalimentadas, possibilitando a persistência de informações. Possuem componentes capazes de decidir quais informações serão mantidas e quais serão esquecidas. A figura 13 apresenta uma célula de LSTM.

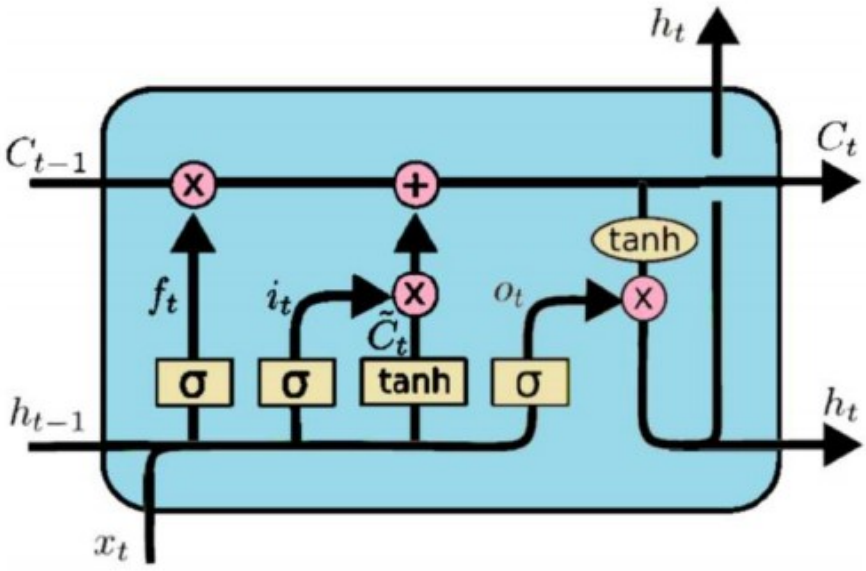

Figura 14 – Representação de uma célula LSTM
Fonte: Adaptada de OLAH, C. 2015

Antes de iniciar o treinamento da LSTM, com o objetivo de facilitar a convergência do algoritmo de aprendizado supervisionado, precisamos transformar a entrada de dados para que o domínio varie entre 0 e 1, aplicando a função MinMaxScaler, definida como:

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)} \qquad (2)$$

```
#Transforma os dados em matriz do tipo coluna
training_set = treinamento.values.reshape(-1,1)

# Feature Scaling
sc = MinMaxScaler(feature_range = (0, 1))
training_set_scaled = sc.fit_transform(training_set)
```

Neste trabalho iremos tirar proveito da capacidade de memória que as Redes Neurais do tipo LSTM possuem. Chamaremos de Janela a quantidade de registros anteriores que serão utilizados no processo de aprendizado de uma LSTM.

A função GeraJanelaTreina cria uma estrutura de dados para o treinamento da LSTM constituída por dois vetores conforme definição a seguir.

$X\_train_i = (X_{i-janela+k}, \ldots, X_{i-1})$ (2)

$Y\_train_i = (X_i)$. (3)

Onde:
k varia de 0 ao tamanho da janela -1
i varia do primeiro elemento da base de treinamento até o último

```
def GeraJanelaTreina (entrada,janela):
    X_train = []
    y_train = []
    for i in range(janela, len(entrada)):
        X_train.append(entrada[i-janela:i, 0])
        y_train.append(entrada[i, 0])
    X_train, y_train = np.array(X_train), np.array(y_train)
    X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
    return (X_train,y_train)
```

A função GeraModelo definida a seguir, cria um modelo a partir da base de treinamento e da quantidade de neurônios nas camadas internas (qtd_neuronios).

```
def GeraModelo (X_train,y_train,qtd_neuronios):
   model = Sequential()

   #cria nivel 1
   model.add(LSTM(units = qtd_neuronios, return_sequences = True, input_shape = (X_train.shape[1], 1)))
   model.add(Dropout(0.2))

   #cria nivel 2
   model.add(LSTM(units = qtd_neuronios, return_sequences = True))
   model.add(Dropout(0.2))

   #cria nivel 3
   model.add(LSTM(units = qtd_neuronios, return_sequences = True))
   model.add(Dropout(0.2))

   #cria nivel 4
   model.add(LSTM(units = qtd_neuronios))
   model.add(Dropout(0.2))

   #cria nivel de saida
   model.add(Dense(units = 1))

   # Compila a RNN
   model.compile(optimizer = 'adam', loss = 'mean_squared_error')

   # Treina a rede
   model.fit(X_train, y_train, epochs = 100, batch_size = 32)
```

Um modelo LSTM é capaz de prever um elemento de uma série a partir de um vetor composto por n elementos anteriores. A função GeraJanelaPrev cria uma estrutura de dados para este fim. O vetor X_test terá o tamanho indicado pela variável Janela e será definido conforme abaixo:

$X\_test_{i} = (X_{i-janela+k,\ldots}, X_{i-1})$, (4)

Onde:

- X representa o vetor com todos os elementos da série temporal
- k varia de 0 ao tamanho da janela -1
- i varia do primeiro elemento da base de teste até o último.

```
def GeraJanelaPrev(total_set_scaled,tamanho_teste,janela):
    inputs = total_set_scaled[len(total_set_scaled) - tamanho_teste - janela:]
    X_test = []
    for i in range(janela, len(inputs)): # antes era 708
        X_test.append(inputs[i-janela:i, 0])
    X_test = np.array(X_test)
    X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
    return(X_test)
```

Por fim, a função aplica_modelo irá chamar as rotinas acima de forma coordenada, executando os seguintes passos:

1. Utiliza a função MinMaxScaler nos dados da série para que eles passem a variar de 0 a 1;
2. Prepara os dados de treinamento (GeraJanelaTreina);
3. Gera o modelo (GeraModelo) com os dados de treinamento;
4. Prepara os dados para a previsão (GeraJanelaPrev);
5. Aplica o modelo gerado no passo 3;
6. Aplica a transformação inversa MinMaxScaler;
7. Avalia a eficiência do modelo comparando com os dados de teste.

```
def aplica_modelo (janela,unidades,total,treinamento,teste):

    training_set = treinamento.values.reshape(-1,1)

    # Faz a transformação dos dados de treinamento para variarem de 0 a 1
    sc = MinMaxScaler(feature_range = (0, 1))
    training_set_scaled = sc.fit_transform(training_set)

    #Prepara dados de treinamento
    X_train,y_train=GeraJanelaTreina(training_set_scaled,janela)

    # Gera modelo
    model=GeraModelo (X_train,y_train,unidades)

    # prepara dados para teste
    total_set=total.values.reshape(-1,1)
    total_set_scaled = sc.transform(total_set)
    X_test=GeraJanelaTeste(total_set_scaled,len(teste),janela)

    # aplica o modelo nos dados de teste
    predicted_scaled = model.predict(X_test)
```

```
    # Aplica a transformação inversa na previsão
    predicted = sc.inverse_transform(predicted_scaled)

    # Avalia resultado
    rmse = float(math.sqrt(mean_squared_error(teste.values, predicted)))
    return (model,sc,rmse)
```

Com o objetivo de escolher os melhores hiperparâmetros foi desenvolvida a função gird_LTSM que gera um modelo para cada um dos conjuntos de parâmetros passados e armazena, em uma tabela, um indicador da qualidade do modelo. O indicador escolhido foi o Root Mean Squared Error (RMSE), quanto menor o seu valor melhor é o modelo.

```
def grid_LTSM(lista_parametros,total,treinamento,teste):
    result=pd.DataFrame({'parametro':[],'RMSE':[]})

    for param in lista_parametros:
        try:
            print ("Gerando modelo LTSM {}".format(param))
            modelo,sc,rmse=aplica_modelo (param[0],param[1],total,treinamento,teste)
        except:
            continue
        print ("RMSE = {}".format(rmse))
        result= result.append({'parametro':param,'RMSE':rmse}, ignore_index=True)

    print(result)
    result = result.sort_values(by='RMSE', ascending=True).reset_index(drop=True)

    return result
```

A tabela 7 apresenta os valores possíveis para cada um do hiperparâmetros que terão seus modelos testados, totalizando 64 diferentes modelos.

Parâmetro	Valores
Tamanho da janela	30,40,50,60,70,80,90,95
Quantidade de neurônios na camada intermediária	20,25,30,35,40,45,50,55

Tabela 7 – Lista de hiperparâmetros

A tabela 8 apresenta os 5 melhores modelos encontrados. O melhor modelo utiliza 40 neurônios na camada intermediária e uma janela com 40 registros.

Tamanho da Janela	Quantidade de neurônios na camada intermediária	RMSE
40	40	91,50
50	30	95,20
60	40	97,20
100	25	97,51
50	40	97,54

Tabela 8 – Melhores modelos

O último passo é aplicar o modelo identificado e apresentar os resultados.

```
# Gera o modelo
janela=40
unidades=40
model,sc,rmse=aplica_modelo (janela,unidades,total,treinamento,teste)

# prepara dados para teste
total_set=total.values.reshape(-1,1)
total_set_scaled = sc.transform(total_set)
X_test=GeraJanelaTeste(total_set_scaled,len(teste),janela)

# aplica o modelo nos dados de teste
predicted_scaled = model.predict(X_test)

# Aplica a transformação inversa na previsão
predicted = sc.inverse_transform(predicted_scaled)

# Exibe os resultados
plt.plot(teste.values, color = 'red', label = 'Real')
plt.plot(predicted, color = 'blue', label = 'Predicted')
plt.title('Comparativo previsão e teste')
plt.xlabel('Data')
plt.ylabel('Quantidade')
plt.legend()
plt.show()
```

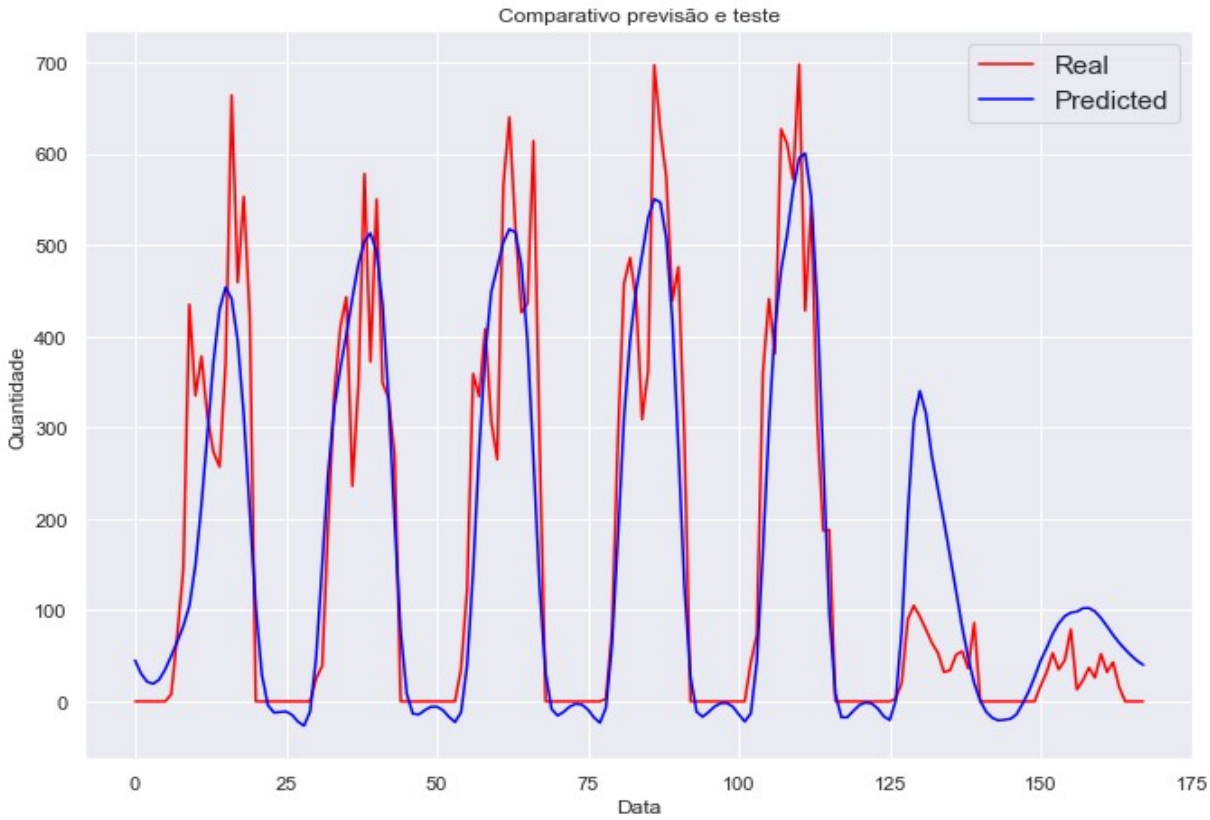

Figura 15 – Comparativo previsão e teste LTSM.

6. Apresentação dos Resultados

Primeiramente, cabe apresentar o workflow motivador deste trabalho seguindo o modelo de Canvas proposto por Vasandani.

Titulo: Previsão da demanda de consultas SQL ao ambiente analítico com o objetivo de subsidiar contratação PaaS em nuvem.		
Problema: Prever a demanda de conslutas SQL no ambiente analítico.	**Resultados:** A previsão do comportamento futuro da quantidade de consultas SQL.	**Aquisição de Dados:** Dados extraídos de duas origens: SGBD Infobright e DW Corporativo conforme detalhado no capítulo 1.
Modelagem: Dois modelos foram testados, LTSM e SARIMA.	**Avaliação dos modelos**: O desempenho dos modelos será avaliado utilizando a raiz da média dos erros quadráticos entre os valores previstos e a base de teste.	**Preparação dos Dados:** Foram realizadas atividades de limpeza nos dados, exploração dos dados e a seleção de atributos, conforme disposto nos capítulos 2 e 3

Tabela 9 – Canvas seguido modelo proposto por Vasandani

Comparando os gráficos das previsões dos dois modelos é possível concluir que visualmente as duas previsões apresentaram resultados satisfatórios:

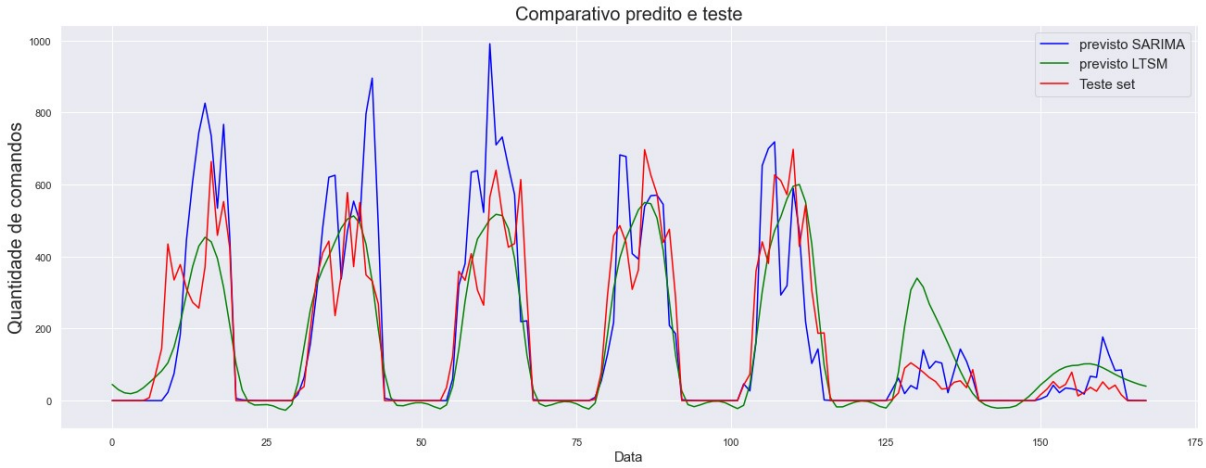

Figura 16 – Comparativo da previsão dos modelos LTSM e SARIMA

A aplicação do estimador RMSE (raiz quadrada da média dos erros quadráticos) nos permite avaliar com maior precisão qual dos modelos possui um menor erro em relação a base de teste.

```
# Gera RMSE para LSTM
rmse = math.sqrt(mean_squared_error(predicted, teste.values))
print('Test RMSE do modelo LSTM: %.3f' % rmse)
```

```
Test RMSE do modelo LSTM: 95.920
```

```
# Gera RMSE para o modelo SARIMA
rmse = math.sqrt(mean_squared_error(forecast1[0:período], teste.values))
print('Test RMSE do modelo SARIMA: %.3f' % rmse)
```

```
Test RMSE do modelo SARIMA: 141.938
```

O modelo LSTM apresenta um RMSE de 95 enquanto que o modelo SARIMA apresenta um RMSE de 142, ou seja, 48% superior. Portanto é possível concluir que, para o caso concreto de prever a demanda de uso do ambiente analítico, o modelo LSTM apresenta melhor capacidade de previsão que o modelo SARIMA.

Importante destacar que o modelo LSTM somente consegue prever o elemento seguinte a partir de 40 elementos anteriores o que dificultaria a realização de previsões de vários elementos futuros. O modelo SARIMA não apresenta esta limitação.

Ambos os modelos criados neste estudo, respeitando as considerações acima, apresentaram resultados satisfatórios e podem ser utilizados para subsidiar o dimensionamento de um serviço analítico na nuvem (PaaS) a ser contratado futuramente.

7. Links

- Link para o repositório Github com o conteúdo do trabalho:

https://github.com/claudiobraga06/TCC.git

- Link para o vídeo com a apresentação resumida do trabalho no Youtube:

https://www.youtube.com/watch?v=467lwdNg2yQ

APÊNDICE

Código integral do programa em Python

```
from pandas import DataFrame
import pandas as pd
import numpy as np
import math

from itertools import product

from datetime import datetime
from datetime import timedelta

import chart_studio.plotly as py
import plotly.express as px
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout
from keras.callbacks import EarlyStopping
from keras.layers import *

from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import train_test_split

import statsmodels.api as sm
from statsmodels.graphics.tsaplots import plot_pacf
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.tsa.holtwinters import ExponentialSmoothing
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.seasonal import seasonal_decompose

from tqdm import tqdm_notebook

import warnings
warnings.filterwarnings('ignore')

#  Carrega tabela CONSULTA e USUÁRIOS
consultas= pd.read_csv('D:\\tcc\\QUERIES_09MAR.csv')
usuarios= pd.read_csv('D:\\tcc\\usuarios 9 mar.csv',dtype={'cpf':str,'orgao':str})
usuarios.rename(columns={'cpf':'cpf_usuario'}, inplace=True)

## Apresenta estrutura das tabelas
consultas.info()
usuarios.info()
```

```
# identifica usuário NAN
ausencia= usuarios.apply(lambda row: row.orgao!=row.orgao, axis=1)
usuarios[ausencia]

# apaga usuário NAN
usuarios.drop(index=76884,inplace=True)

## Corrige o campo USER da tabela CONSULTAS

# O campo USUARIO algumas vezes apresenta o sufixo '@DATALAKE.SERPRO' após o nome
#  do usuário. A  função abaixo corrige este comportamento
def corrige_usuario(row):
    resp=row.user
    if '@DATALAKE.SERPRO' in row.user:
        resp=row.user[:-16]
    return(resp)

consultas['cpf_usuario']=consultas.apply(corrige_usuario,axis=1)

## Inclusão do usuário “Contagil” na tabela USUARIOS
usuarios=usuarios.append({'cpf_usuario':'contagil','orgao':'RF'},ignore_index=True)

# Exibe os usuários agrupados
usuarios.groupby('orgao').count()

## Exclusão das consultas não finalizadas
QueryState_qtd= consultas['queryState'].value_counts()
plot=QueryState_qtd.plot.bar(title="Distribuição de QueryState")

# Exibe o conteúdo
QueryState_qtd

qs_total=sum (QueryState_qtd)
qs_finished=QueryState_qtd['FINISHED']
qs_exception=QueryState_qtd['EXCEPTION']

print ((" {0:.2f}% são concluídos (finished) e {1:.2f}% são erros (exception)."+
       " Os demais estados correspondem {2:.2f}%").format(qs_finished/qs_total*100,
                                          qs_exception/qs_total*100,
                                          100-(qs_finished+qs_exception)/qs_total*100))

consultas=consultas.loc[(consultas.queryState !="EXCEPTION") & (consultas.queryState !="LOST"
),['cpf_usuario','query_id','data_hora']].copy().reset_index(drop=True)

# exibe informações de consultas
consultas.info()

## Correção do campo Data_hora - O campo INICIO apresenta UMA hora a mais que o real

# conversão do tipo do campo Data_hora de String para o Time
def str2time(val):
    try:
        return datetime.strptime(val,'%Y-%m-%d %H:%M:%S')
    except:
        return pd.NaT
consultas['inicio']= consultas.apply(lambda x: str2time(x.data_hora),axis=1)

# Reduão de 1 hora
consultas.loc[:,'inicio'] = consultas.inicio - timedelta(minutes=60)
```

```
# exclusão do campo data_hora
consultas.drop(columns=['data_hora'],inplace=True)

##  Junção das tabelas USUARIOS e CONSULTAS

# mostra a estrutura das tabelas
consultas.info()
usuarios.info()

# Left join.
juncao=pd.merge(consultas,usuarios,how="left")

# Seleciona registros onde orgao é NAN, ou seja, não CPF não foi encontrado na tabela usuarios
juncao['desconhecido']= juncao.apply(lambda row: row.orgao!=row.orgao, axis=1)

# Mostra relação de CPF que não foram encontrados na tabela usuarios
juncao.loc[juncao.desconhecido,'cpf_usuario'].unique()

# Atribui "prestadorSV" a orgão nos casos onde o CPF das consultas não foram encontrados na
tabela usuários.
juncao.loc[juncao.desconhecido,'orgao']="prestadorSV"
juncao.drop(columns=['desconhecido'],inplace=True)

juncao.orgao.value_counts()

dados_pre=juncao.loc[(juncao.orgao!="prestadorSV"),['query_id','inicio']].reset_index(drop=True)

# Geração da base de dados

#DEFINIÇÃO DE PERÍODO
qtd_partes_1hora=1
delta=timedelta(minutes=60/qtd_partes_1hora)

periodo_dia=24*qtd_partes_1hora
periodo_semana=7*periodo_dia

inicioAnalise=datetime.strptime('2021-02-01 00:00:00','%Y-%m-%d %H:%M:%S')

# Pega o último tempo da serie
fimAnalise=str(dados_pre.sort_values('inicio').inicio.tail(1).values[0])
fimAnalise=fimAnalise[0:10]+" "+fimAnalise[11:19]
fimAnalise=datetime.strptime(fimAnalise,'%Y-%m-%d %H:%M:%S')
print (inicioAnalise,"<->",fimAnalise)

# Gera nova base
dados=pd.DataFrame(
   {'Tempo':[],'qtd':[]})

momento=inicioAnalise

while momento<fimAnalise:
   qtd=dados_pre[(dados_pre.inicio >= (momento)) &
           (dados_pre.inicio <  (momento+delta))]['query_id'].count()
   dados = dados.append({'Tempo':momento,'qtd':qtd}, ignore_index=True)
   print (momento,'qtd=',qtd)
   momento+=delta
```

```
# Análise e exploração dos dados

dados.info()

periodo=periodo_semana
qtd_periodo=len(dados)//periodo
ultimo_periodo=qtd_periodo*periodo

# Mostra a utilização do ambiente analítico

sns.set(rc={'figure.figsize':(11, 8)})
plt.plot(dados.loc[:,'Tempo'],dados.loc[:,'qtd'], color = 'blue', label = 'Total SQL')

plt.title('Utilização do ambiente analítico', fontsize=20)
plt.xlabel('Data', fontsize=20)
plt.ylabel('Quantidade de comandos SQL', fontsize=15)
params = {'legend.fontsize': 15}
plt.rcParams.update(params)
plt.legend()
plt.show()

sns.set(rc={'figure.figsize':(5, 4)})
fig, axs = plt.subplots(qtd_periodo)
fig.suptitle('Gráfico Semanal')

for loop in range(qtd_periodo):
    axs[loop].plot(dados.qtd[periodo*loop:periodo*(loop+1)])
    axs[loop].grid(False)
    axs[loop].set_yticks([])
    axs[loop].set_xticks([])

# Exclui o período do carnaval
antes=dados[:periodo*2]
depois=dados[periodo*3:qtd_periodo*periodo]
dados=antes.append(depois).reset_index(drop=True)
qtd_periodo-=1

# Mostra a utilização do ambiente analítico
sns.set(rc={'figure.figsize':(11, 8)})
plt.plot(dados.loc[:,'qtd'].values, color = 'blue', label = 'Total SQL')
plt.title('Utilização do ambiente analítico sem Carnaval', fontsize=20)
plt.xlabel('Data', fontsize=15)
#plt.xticks(rotation=90)
plt.ylabel('Quantidade de comandos SQL', fontsize=15)
params = {'legend.fontsize': 15}
plt.rcParams.update(params)
plt.legend()
plt.show()

##  Segmentação da base de dados

#Segmenta os dados
total=dados.loc[:ultimo_periodo-1,'qtd'] # 3 semanas
treinamento = dados.loc[:((qtd_periodo-1)*periodo-1), 'qtd'] # 2 semanas
teste= dados.loc[(qtd_periodo-1)*periodo:(qtd_periodo)*periodo-1, 'qtd'] # 1 semana

plt.plot(treinamento, color = 'red', label = 'treinamento')
plt.plot(teste, color = 'blue', label = 'teste')
plt.title('Segmentação da base de dados',fontsize=20)
plt.xlabel('Data',fontsize=15)
```

```
plt.ylabel('Tamanho na fila',fontsize=15)
params = {'legend.fontsize': 15}
plt.rcParams.update(params)
plt.legend()
plt.show()

# TECNICAS DE PREVISÃO DE SERIES TEMPORAIS

# 1) LSTM

def GeraJanelaTreina (entrada,janela):
    X_train = []
    y_train = []
    for i in range(janela, len(entrada)):
        X_train.append(entrada[i-janela:i, 0])
        y_train.append(entrada[i, 0])
    X_train, y_train = np.array(X_train), np.array(y_train)
    X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
    return (X_train,y_train)


def GeraModelo (X_train,y_train,un):
    model = Sequential()

    #cria nivel 1
    model.add(LSTM(units = un, return_sequences = True, input_shape = (X_train.shape[1], 1)))
    model.add(Dropout(0.2))

    #cria nivel 2
    model.add(LSTM(units = un, return_sequences = True))
    model.add(Dropout(0.2))

    #cria nivel 3
    model.add(LSTM(units = un, return_sequences = True))
    model.add(Dropout(0.2))

    #cria nivel 4
    model.add(LSTM(units = un))
    model.add(Dropout(0.2))

    #cria nivel de saida
    model.add(Dense(units = 1))

    # Compila a RNN
    model.compile(optimizer = 'adam', loss = 'mean_squared_error')

    # Treina a rede
    model.fit(X_train, y_train, epochs = 100, batch_size = 32)
    return (model)

def GeraJanelaTeste(total_set_scaled,tamanho_teste,janela):
    inputs = total_set_scaled[len(total_set_scaled) - tamanho_teste - janela:]
    X_test = []
    for i in range(janela, len(inputs)): # antes era 708
        X_test.append(inputs[i-janela:i, 0])
    X_test = np.array(X_test)
    X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
    return(X_test)
```

```
def aplica_modelo (janela,unidades,total,treinamento,teste):

    training_set = treinamento.values.reshape(-1,1)

    # Faz a transformação dos dados de treinamento para variarem de 0 a 1
    sc = MinMaxScaler(feature_range = (0, 1))
    training_set_scaled = sc.fit_transform(training_set)

    #Prepara dados de treinamento
    X_train,y_train=GeraJanelaTreina(training_set_scaled,janela)

    # Gera modelo
    model=GeraModelo (X_train,y_train,unidades)

    # prepara dados para teste
    total_set=total.values.reshape(-1,1)
    total_set_scaled = sc.transform(total_set)
    X_test=GeraJanelaTeste(total_set_scaled,len(teste),janela)

    # aplica o modelo nos dados de teste
    predicted_scaled = model.predict(X_test)

    # Aplica a transformação inversa na previsão
    predicted = sc.inverse_transform(predicted_scaled)

    # evaluate forecasts
    rmse = float(math.sqrt(mean_squared_error(teste.values, predicted)))
    return (model,sc,rmse)

def grid_LTSM(lista_parametros,total,treinamento,teste):

    result=pd.DataFrame({'parametro':[],'RMSE':[]})

    for param in lista_parametros:

        try:
            print ("Gerando modelo LTSM {}".format(param))
            modelo,sc,rmse=aplica_modelo (param[0],param[1],total,treinamento,teste)
        except:
            continue
        print ("RMSE = {}".format(rmse))
        result= result.append({'parametro':param,'RMSE':rmse}, ignore_index=True)

    print(result)
    result = result.sort_values(by='RMSE', ascending=True).reset_index(drop=True)

    return result

# Roda um grid com 256 possibilidades distinatas para procurar o SARIMA com melhor AIC
jan = [40,50,60,100,168]
unid = [20,25,30,40]
jan = [40]
unid = [20,25,30,40]

parameters = product(jan,unid)
parameters_list = list(parameters)

processa=False
if processa:
    #parameters_list=[(20,20)]
```

```
    result_df11 = grid_LTSM(parameters_list,total,treinamento,teste)
else:
    results=[[(40, 40),91.480931],[(50, 30),95.203544],[(60, 40),97.208516],[(100, 25),97.517014],
            [(50, 40),97.543047]]
    result_df11 =pd.DataFrame(results)
    result_df11.columns = ['parametro','RMSE']
    result_df11 = result_df11.sort_values(by='RMSE', ascending=True).reset_index(drop=True)

# apresenta os melhores
result_df11.head(5)

# Gera o modelo
janela=40
unidades=40
model,sc,rmse=aplica_modelo (janela,unidades,total,treinamento,teste)

# prepara dados para teste
total_set=total.values.reshape(-1,1)
total_set_scaled = sc.transform(total_set)
X_test=GeraJanelaTeste(total_set_scaled,len(teste),janela)

# aplica o modelo nos dados de teste
predicted_scaled = model.predict(X_test)

# Aplica a transformação inversa na previsão
predicted = sc.inverse_transform(predicted_scaled)

# Exibe os resultados
plt.plot(teste.values, color = 'red', label = 'Real')
plt.plot(predicted, color = 'blue', label = 'Predicted')
plt.title('Comparativo previsão e teste')
plt.xlabel('Data')
plt.ylabel('Quantidade')
plt.legend()
plt.show()

# calcula RMSE
rmse = math.sqrt(mean_squared_error(predicted, teste.values))
print('Test RMSE da LSTM: %.3f' % rmse)

#CRIA MODELO SARIMA

# Decompõe a Serie
result = seasonal_decompose(treinamento.values,period=periodo_semana)

fig = result.plot()

plot_pacf(treinamento);
plot_acf(treinamento);

# AR=1 (PACF tem um pico significante em 1 e outro em 7) ->P=1

# p-value < 0,05 -> Rejeitada a hipótse nula. Série é estacionária.
ad_fuller_result = adfuller(treinamento)
print(f'ADF Statistic: {ad_fuller_result[0]}')
print(f'p-value: {ad_fuller_result[1]}')

treinamento_aj = treinamento.diff(periodo)
treinamento_aj = treinamento_aj[periodo:].reset_index(drop=True)
```

```
ad_fuller_result = adfuller(treinamento_aj)
print('ADF Statistic:{} ,p-value:{}'.format(ad_fuller_result[0],ad_fuller_result[1]))
treinamento_aj.plot()

plot_pacf(treinamento_aj);
plot_acf(treinamento_aj);

def grid_SARIMA(lista_parametros, s, exog):
    aic=0
    result=pd.DataFrame({'(p,d,q)x(P,D,Q)':[],'AIC':[]})

    for param in lista_parametros:

        try:
            print ("Gerando modelo SARIMAX ({},{},{})({},{},{}){}".format(param[0], param[1],
param[2],param[3], param[4], param[5], s))
            model = SARIMAX(exog, order=(param[0], param[1], param[2]), seasonal_order=(param[3],
param[4], param[5], s)).fit(disp=-1)
        except:
            continue

        aic = model.aic
        print ("AIC encontrado={}".format(aic))
        result= result.append({'(p,d,q)x(P,D,Q)':param,'AIC':qtd}, ignore_index=True)

    result = result.sort_values(by='AIC', ascending=True).reset_index(drop=True)

    return result

# Roda um grid com 256 possibilidades distinatas para procurar o SARIMA com melhor AIC
p = range(0, 4, 1)
d = range(1,2,1)
q = range(0, 4, 1)
P = range(0, 4, 1)
D = range (1,2,1)
Q = range(0, 4, 1)
s = periodo
parameters = product(p, d, q, P,D,Q)
parameters_list = list(parameters)

result_df11 = grid_SARIMA(parameters_list, periodo, treinamento.values)

result_df11.head(5)

model = SARIMAX(treinamento.values, order=(2, 1, 2), seasonal_order=(1, 1, 0, periodo)).fit()
print(model.summary())

# Faz a previsão
forecast1 = model.predict()

# gera o gráfico comparando os resultados LTSM e SARIMA
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(rc={'figure.figsize':(22, 8)})

plt.plot(forecast1[0:periodo], color = 'blue', label = 'previsto SARIMA')
plt.plot(predicted, color = 'green', label = 'previsto LTSM')

plt.plot(teste.values, color = 'red', label = 'Teste set')
#plt.xticks(np.arange(0,587,50))
```

```
plt.title('Comparativo predito e teste',fontsize=20)
plt.xlabel('Data',fontsize=15)
plt.ylabel('Quantidade de comandos',fontsize=20)
params = {'legend.fontsize': 15}
plt.rcParams.update(params)
plt.legend()
plt.show()

# Gera RMSE para SARIMA
rmse = math.sqrt(mean_squared_error(forecast1[0:periodo], teste.values))
print('Test RMSE do modelo SARIMA: %.3f' % rmse)
```