WEB ANALYTICS: MODELANDO SÉRIE TEMPORAL COM ARIMA E MLP

Autora: Maria Francinete Mateus

Objetivo: Analisar o padrão de acessos ao blog *Viagem de Cinema* e identificar qual o modelo funciona melhor para fazer predições com eles: o *Auto-Regressive Integrated Moving Average* (ARIMA) ou o *Multilayer Perceptron* (MLP).

Pacotes e funções necessários

In [1]:

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
from matplotlib.dates import DateFormatter
from statsmodels.tsa.seasonal import seasonal_decompose

from scipy import stats
import statsmodels.api as sm
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.stats.diagnostic import acorr_ljungbox
from pmdarima.arima import auto_arima

from sklearn.neural_network import MLPRegressor
from sklearn.model_selection import TimeSeriesSplit
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler
import warnings
warnings.filterwarnings('ignore')
```

Coleta e exploração dos dados

Base de dados: O blog *Viagem de Cinema* (https://viagemdecinema.blogspot.com) publica conteúdo sobre locações de filmes e séries de tevê desde maio de 2010. Ele foi construído usando Blogger, uma plataforma de criação, edição e gestão de blogs de propriedade do Google. Os dados para esse estudo foram coletados diretamente das estatísticas fornecidas por essa plataforma.

Período da série: 01/06/2010 - 31/05/2022

In [2]:

```
VC = pd.read_csv('dados/VC-dados-total.csv', parse_dates=['Meses'], index_col=['Meses'])
VC.head()
```

Out[2]:

	#Acessos
Meses	
2010-06-01	33
2010-07-01	244
2010-08-01	166
2010-09-01	196
2010-10-01	347

In [3]:

```
VC.describe()
```

Out[3]:

	#Acessos
count	144.000000
mean	7313.875000
std	6147.324646
min	33.000000
25%	3776.750000
50%	5506.000000
75%	8627.500000
max	38726.000000

Comentário: *A quantidade de acessos mensais do período analisado apresentou o mínimo de 33 acessos, em 06/2010, e o máximo de 38.726 acessos, em 08/2016. A média mensal de acessos foi de 7.314, quantidade superior ao da mediana, que foi de 5.506 acessos.*

DECOMPOSIÇÃO DA SÉRIE TEMPORAL

A técnica de *decomposição* será utilizada para podermos examinar tanto a distribuição dos dados do blog ao longo do tempo como se eles mostram algum padrão de:

- **tendência** geral (*trend*), através da captura de mudanças no nível ao longo do tempo;
- **sazonalidade** (*seasonal*), através da captura de efeitos cíclicos de acordo com a época do ano; e
- **resíduos** (*residuals*) ou comportamentos irregulares não descritos pelos efeitos de tendência ou de sazonalidade.

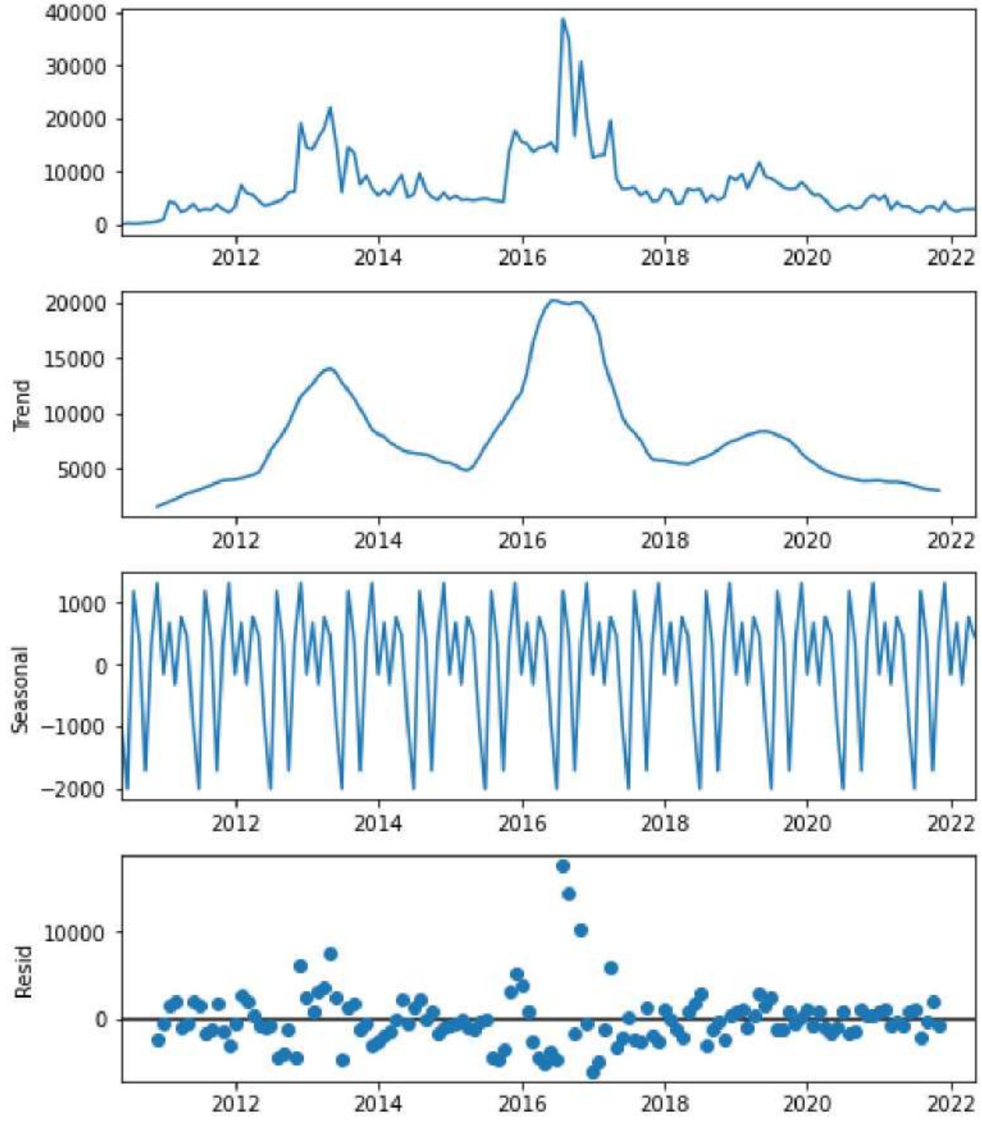
A decomposição dos dados pode ser *aditiva* ou *multiplicativa*.

Modelo Aditivo

Neste modelo, os componentes são somados para gerar os valores da série temporal e os resíduos são centralizados em torno de 0.

In [4]:

```
plt.rcParams['figure.figsize'] = [7, 8]
sd = seasonal_decompose(VC, period=12).plot()
```



Comentários:

- *O primeiro gráfico mostra a distribuição dos dados de acessos ao blog Viagem de Cinema ao longo dos 12 anos analisados.*

- *O gráfico **Trend** indica que não existe uma tendência geral de acessos ao blog, mas flutuações aleatórias ao longo dos anos, com altas em 2013, pico em agosto de 2016 e queda contínua a partir de maio/2017. Em 2019, houve um leve incremento nos acessos, mas, desde então, os dados se* *estabilizaram numa média mensal de 3.500 acessos. Ou seja, trata-se de uma série temporal com dados complexos, que demandam técnicas avançadas como ARIMA e MLP.*

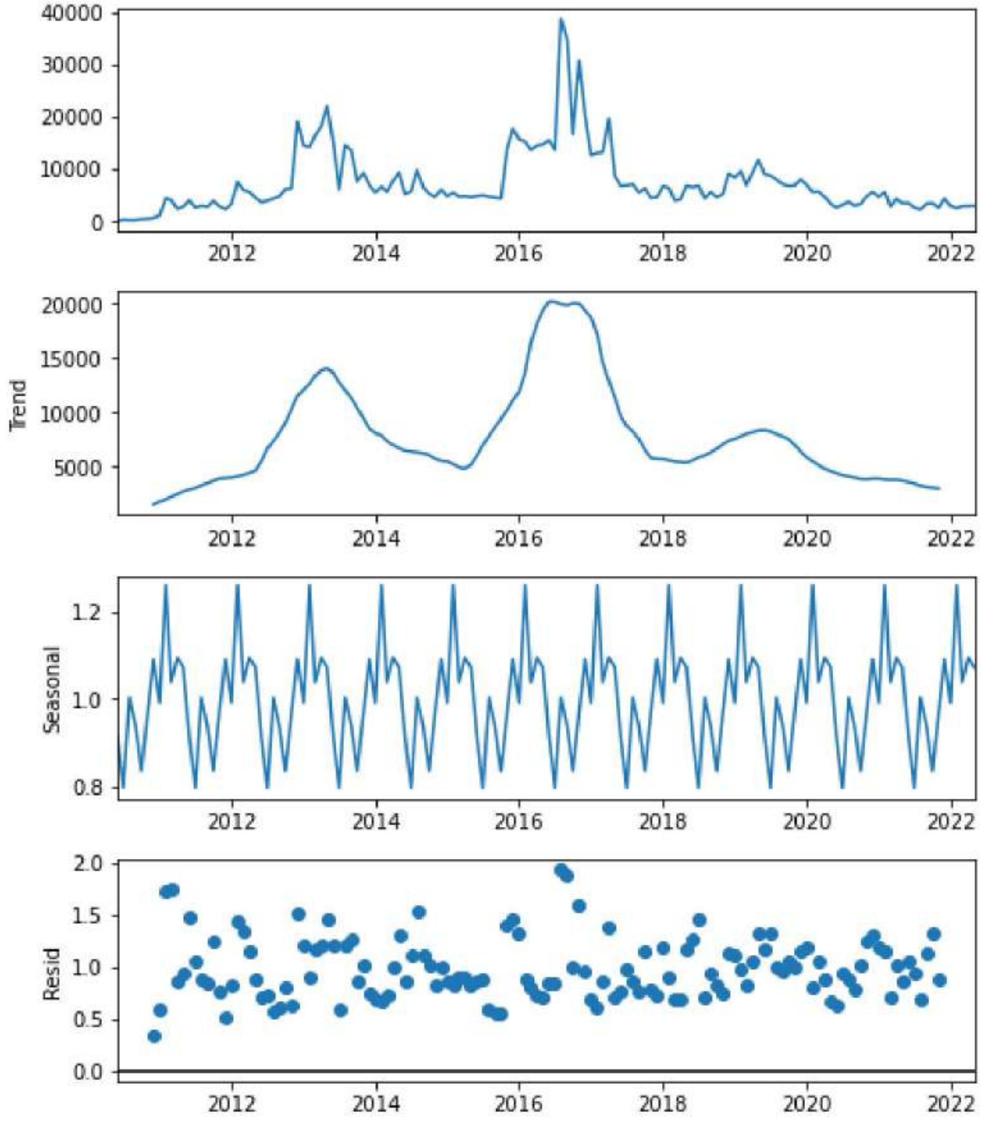
- *O gráfico **Seasonal** mostra que existe um componente de sazonalidade consistente, que se repete todos os anos.*
- *O gráfico **Resid** mostra as irregularidades existentes nos dados, especialmente os outliers existentes entre 08 e 11/2016.*

Modelo Multiplicativo

Neste modelo, os componentes são multiplicados para gerar os valores da série temporal e os resíduos são centralizados em torno de 1.

In [5]:

```
sd = seasonal_decompose(VC, model='multiplicative').plot()
```



Comentários:

As principais diferenças entre esse modelo e o aditivo encontram-se nos gráficos de sazonalidade e resíduos:

- *No de sazonalidade, com os dados distribuídos em coeficientes entre 0.8 e +1.2, pode-se ver como os padrões tanto de alta como de baixas nos acessos são similares em todos os anos, o que pode ser útil durante o processo de modelagem para previsão de acessos futuros.*

- *O gráfico de resíduos mostra que, independente da mudança na escala de apresentação (coeficientes entre 0.0 e 2.0 e centralização em 1.0), as irregularidades se apresentam em todo o período analisado. O ACF (Função de Autocorrelação) e o teste Ljung-Box mostrarão se esses componentes de irregularidades têm ou não significância estatística.*

MODELO 1: ARIMA

Separação dos dados em Treino e Teste

- Treino: dados do período de 06/2010 até 05/2019
- Teste: dados do período de 06/2019 até 05/2022

In [6]:
```
trn = VC.loc[VC.index < '2019-06-01']

tst = VC.loc[VC.index >= '2019-06-01']
```

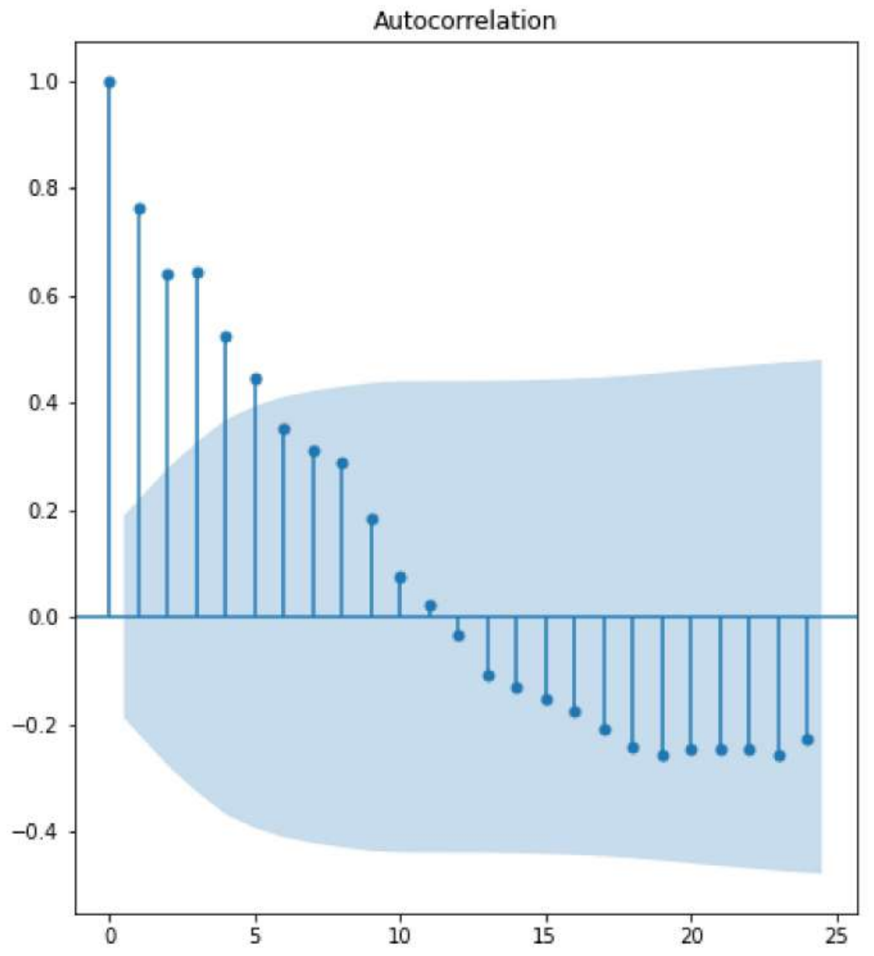
Preparação dos dados para aplicação do ARIMA

- O ARIMA requer dados não estacionários, em que média, variância e/ou covariância variam ao longo do tempo.
- Os dados não estacionários mostram correlações significativas quando são defasados; *quanto mais próximos de 0, melhor*.
- O teste de estacionaridade da série será feito usando Função de Autocorrelação (ACF).

Usando **ACF** (*Autocorrelation Function* ou Função de Autocorrelação)

- ACF é uma medida da correlação entre as observações de uma série temporal separadas por k unidades de tempo chamadas de *lag*, mostrando como elas se autocorrelacionam quando são defasadas.
- O eixo horizontal indicará a defasagem dos dados da série (que será de 24 meses ou lags).
- O eixo vertical indicará a autocorrelação entre os lags (lembrando que, quanto mais próximo de 0, melhor para o ARIMA).

In [7]:
```
trn_acf = plot_acf(trn, lags=24)
```



Comentários:

- *Os valores dos lags 1 ao 6 (no espaço branco) são altamente correlacionados.*
- *Há um pico no lag 1 que diminui ao longo de vários períodos até o lag 19. Nota-se que, mesmo quando voltam a subir, os valores permanecem dentro dos limites da faixa azul, onde a autocorrelação é igual ou próxima de 0, significando que essa série é aleatória, de dados não estacionários, como é esperado pelo ARIMA.*

Usando **Regressão Linear** como base de comparação para o ARIMA

- A variável dependente é o número de acessos.

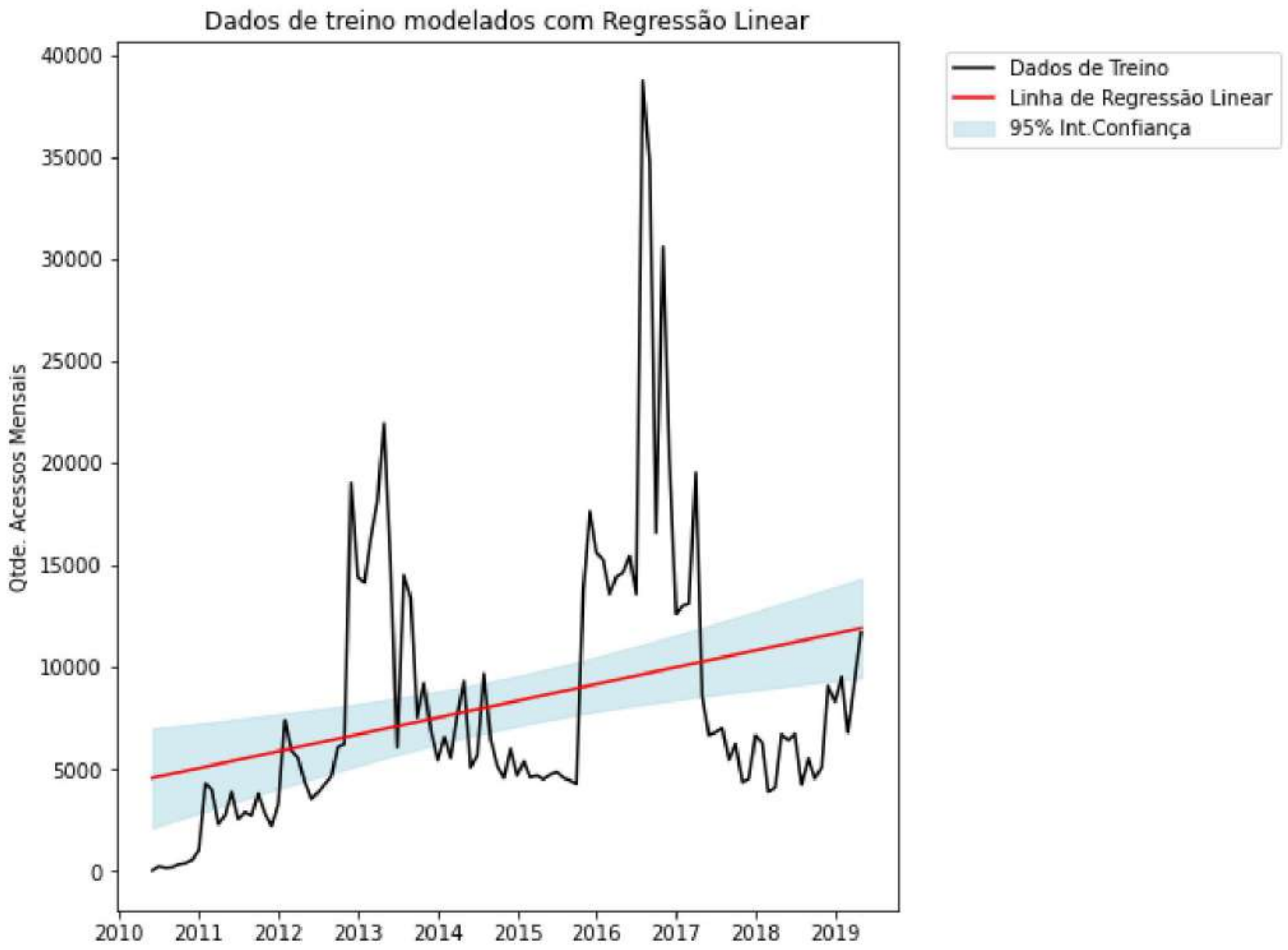
- A variável independente será uma tendência temporal linear.
- O intervalo de confiança será de 95%.

In [8]:

```
lr = sm.OLS(endog=trn['#Acessos'], exog=sm.add_constant(np.arange(1, 1 + trn.shape[0]))).fit()

y_hat = lr.fittedvalues

y_ci = lr.get_prediction().conf_int(alpha=0.05)
```

In [9]:

```
fig, ax = plt.subplots()
plt.ylabel('Qtde. Acessos Mensais')
plt.title('Dados de treino modelados com Regressão Linear')
plt.plot(trn, color='black', label='Dados de Treino')
plt.plot(y_hat, color='red', label='Linha de Regressão Linear')
plt.fill_between(y_hat.index, y_ci[:, 0], y_ci[:, 1], color='lightblue', alpha=0.5, label='95% Int.Confiança')
plt.legend(bbox_to_anchor=(1.05, 1))
ax.xaxis.set_major_formatter(DateFormatter('%Y'))
```



Comentário: *A linha de regressão linear (em vermelho) e o intervalo de confiança (em azul) mostram que as sazonalidades da série não estão sendo capturadas por esse modelo. Agora, usaremos o ARIMA para ver se ele consegue capturar isso.*

Treinamento dos dados com o ARIMA

- Os parâmetros ótimos do modelo serão encontrados usando a função `auto_arima` do pacote `pmdarima` .
- Por padrão, o período de sazonalidade 'm' é igual a 1. Usando m=12 assegura-se que o modelo levará em consideração a sazonalidade anual observada na série temporal (ou seja, um ciclo a cada 12 meses).
- O modelo será construído com o algoritmo SARIMAX (*Seasonal Auto-Regressive Integrated Moving Average with Exogenous Factors*), uma variação do ARIMA.

In [10]:

```
auto_arima_model = auto_arima(trn, m=12, with_intercept=False, suppress_warnings=True)
auto_arima_model.order
```

Out[10]:

```
(3, 0, 1)
```

Os parâmetros ótimos do modelo são 3, 0, 1, sendo:

- 3 = p: Auto-regressive (AR)
- 0 = d: Integrate (I)
- 1 = q: Moving average (MA)

In [11]:

```
auto_arima_model.seasonal_order
```

Out[11]:

```
(0, 0, 0, 12)
```

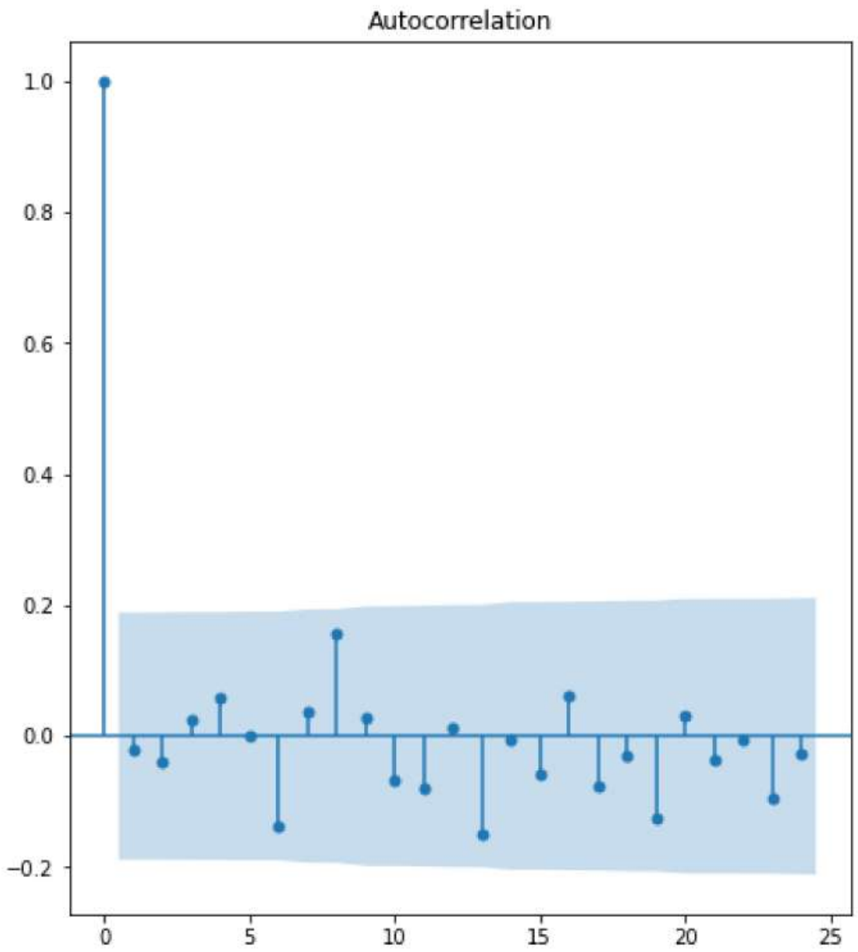
Os padrões sazonais ótimos são 0, 0, 0, 12, sendo:

- 0 = P: Auto-regressive (AR)
- 0 = D: Integrate (I)
- 0 = Q: Moving average (MA)
- 12= M: Sazonalidade

Aplicando o Auto_Arima e ACF sobre os dados resíduals

In [12]:

```
resid = auto_arima_model.resid()

res_acf = plot_acf(resid, lags=24)
```



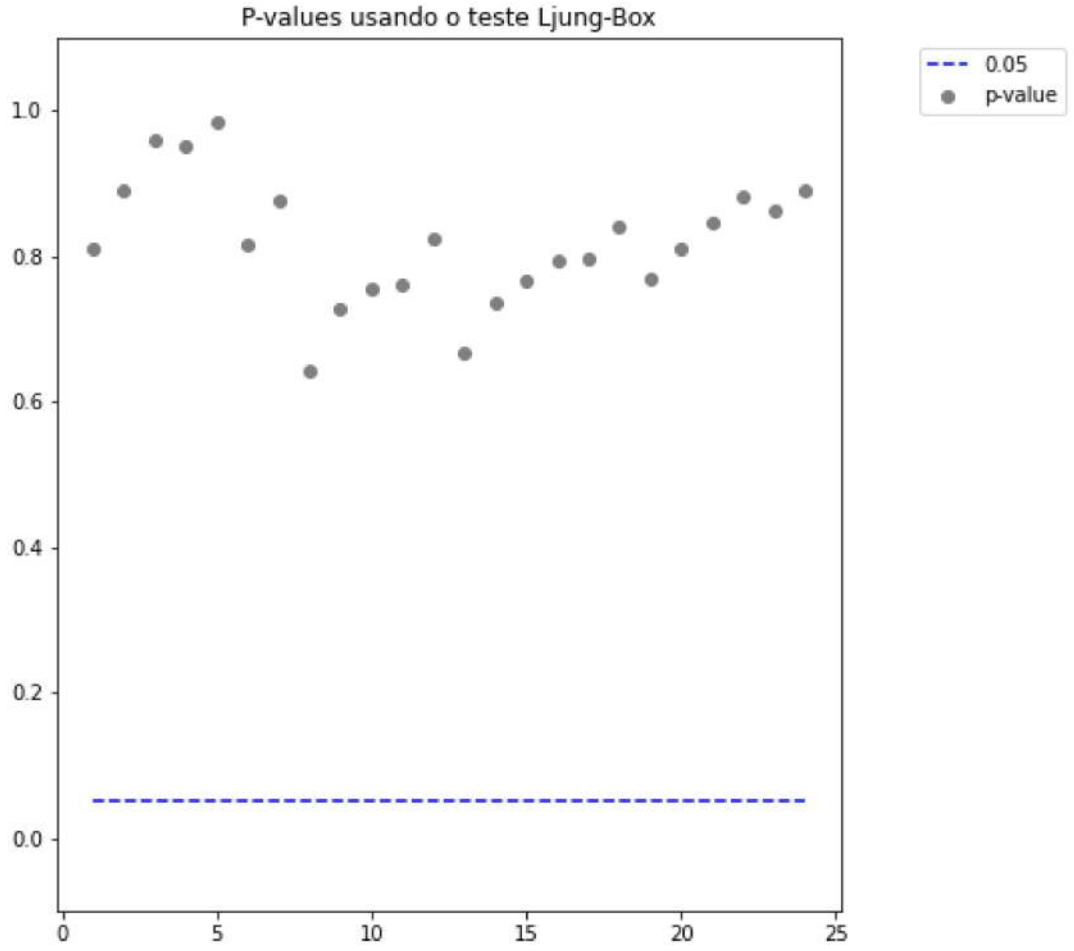
Comentário: *Depois do primeiro mês, os resíduos cairam na faixa azul, região de não significância estatística.*

Aplicando o teste Ljung-Box sobre os dados resíduais

In [13]:

```
res_lb = acorr_ljungbox(resid, lags=24, return_df=False)[1]

fig, ax = plt.subplots()
plt.title('P-values usando o teste Ljung-Box')
plt.scatter(np.arange(1, 1 + len(res_lb)), res_lb, color='gray', label='p-value')
plt.plot(np.arange(1, 1 + len(res_lb)), [0.05] * len(res_lb), '--', color='blue', label='0.05')
plt.ylim(-0.1, 1.1)
plt.legend(bbox_to_anchor=(1.3, 1))
```

Out[13]: <matplotlib.legend.Legend at 0x293bc228d00>



Comentário: *Os p-values dos valores residuais estão muito acima de 0.05, ou seja, não têm significância estatística, que é o que estamos procurando aqui.*

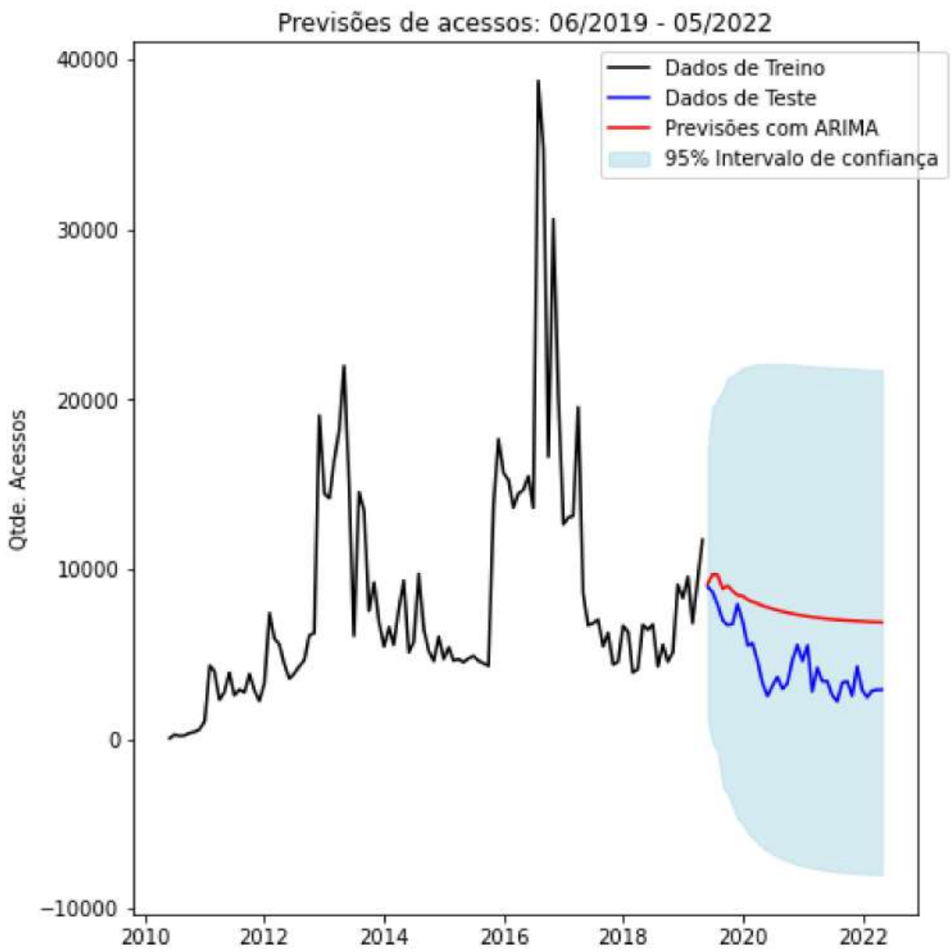
Construindo o modelo com os melhores parâmetros

In [14]:
```
auto_arima_model.fit(trn)

arima_predictions = auto_arima_model.predict(n_periods=36, alpha=0.05, return_conf_int=True)
```

```
y_pred = pd.Series(arima_predictions[0], index=tst.index)

y_pred_lb, y_pred_ub = arima_predictions[1][:, 0], arima_predictions[1][:, 1]
```

In [15]:

```
fig, ax = plt.subplots()
plt.ylabel('Qtde. Acessos')
plt.title('Previsões de acessos: 06/2019 - 05/2022')
plt.fill_between(tst.index, y_pred_lb, y_pred_ub, color='lightblue', alpha=0.5, label='95% Intervalo de confiança')
plt.plot(trn, color='black', label='Dados de Treino')
plt.plot(tst, color='blue', label='Dados de Teste')
plt.plot(y_pred, color='red', label='Previsões com ARIMA')
plt.legend(bbox_to_anchor=(1.05, 1))
ax.xaxis.set_major_formatter(DateFormatter('%Y'))
```



Comentário: *O modelo criado com o ARIMA mostrou que tem capacidade de tratar dados complexos, apesar das previsões serem superiores ao realizado.*

Valores previstos dentro dos intervalos de confiança

In [16]:

```
tst_pred = pd.DataFrame({
    'Realizado': tst.iloc[:, 0].values,
    'Previsto': y_pred.values,
    'Abaixo 95': y_pred_lb,
    'Acima 95': y_pred_ub
    }, index=tst.index)

tst_pred.head(12)
```

Out[16]:

Meses	Realizado	Previsto	Abaixo 95	Acima 95
2019-06-01	8956	9126.381595	1097.882722	17154.880467
2019-07-01	8674	9665.878030	-234.507950	19566.264010
2019-08-01	7916	9646.568037	-744.920693	20038.056767
2019-09-01	6986	8843.001255	-2849.037105	20535.039615
2019-10-01	6703	9021.178232	-3205.916020	21248.272484
2019-11-01	6774	8720.115924	-3950.255510	21390.487357
2019-12-01	7941	8466.058512	-4680.308526	21612.425549
2020-01-01	6931	8415.647924	-5010.262467	21841.558316
2020-02-01	5506	8202.997925	-5486.154901	21892.150752
2020-03-01	5630	8079.508913	-5827.346082	21986.363909
2020-04-01	4591	7977.242474	-6090.609644	22045.094591
2020-05-01	3300	7845.883561	-6364.784130	22056.551252

Comentário: *Os valores acima mostram os valores previstos, bem como as estimativas de valores mínimos e valores máximos mensais. Para que tais valores pudessem ser alcançados, a gestão do blog teria que considerar investimentos em campanhas de marketing simultâneas em diferentes mídias tal como fez nos meses de picos de 2016 e 2017.*

MODELO 2: MLP

Tratamento e separação dos dados

In [17]:

```
k = 12

Z = []

for i in range(k + 1, VC.shape[0] + 1):
    Z.append(VC.iloc[(i - k - 1): i, 0])
```

```
Z = np.array(Z)

Z.shape
```

Out[17]:

```
(132, 13)
```

Dados de Treino e de Teste

- Treino: 80% dos dados
- Teste: 20% dos dados

In [18]:

```
split = np.int(0.80 * Z.shape[0])
Z_train, Z_test = Z[:split, :], Z[split:, :]
```

In [19]:

```
scaler = StandardScaler().fit(Z_train)
Z_train = scaler.transform(Z_train)
Z_test = scaler.transform(Z_test)

X_train, y_train = Z_train[:, :-1], Z_train[:, -1]
X_test, y_test = Z_test[:, :-1], Z_test[:, -1]
```

Treinamento dos dados e identificação do melhor nº de hidden nodes

In [20]:

```
split_ = np.int(0.80 * X_train.shape[0])
X_train_, y_train_ = X_train[:split_, :], y_train[:split_]
X_valid_, y_valid_ =  X_train[split_:, :], y_train[split_:]

def validation_loss(hidden_neurons):

    mlp = MLPRegressor(hidden_layer_sizes=(hidden_neurons,), max_iter=500, random_state=1, shuffle=False)

    mlp.fit(X_train_, y_train_)

    return mean_squared_error(y_valid_, mlp.predict(X_valid_))

params = [5, 10, 25, 50, 75, 100]
```

```
mse = [validation_loss(p) for p in params]

params[np.argmin(mse)]
```

Out[20]:

```
5
```

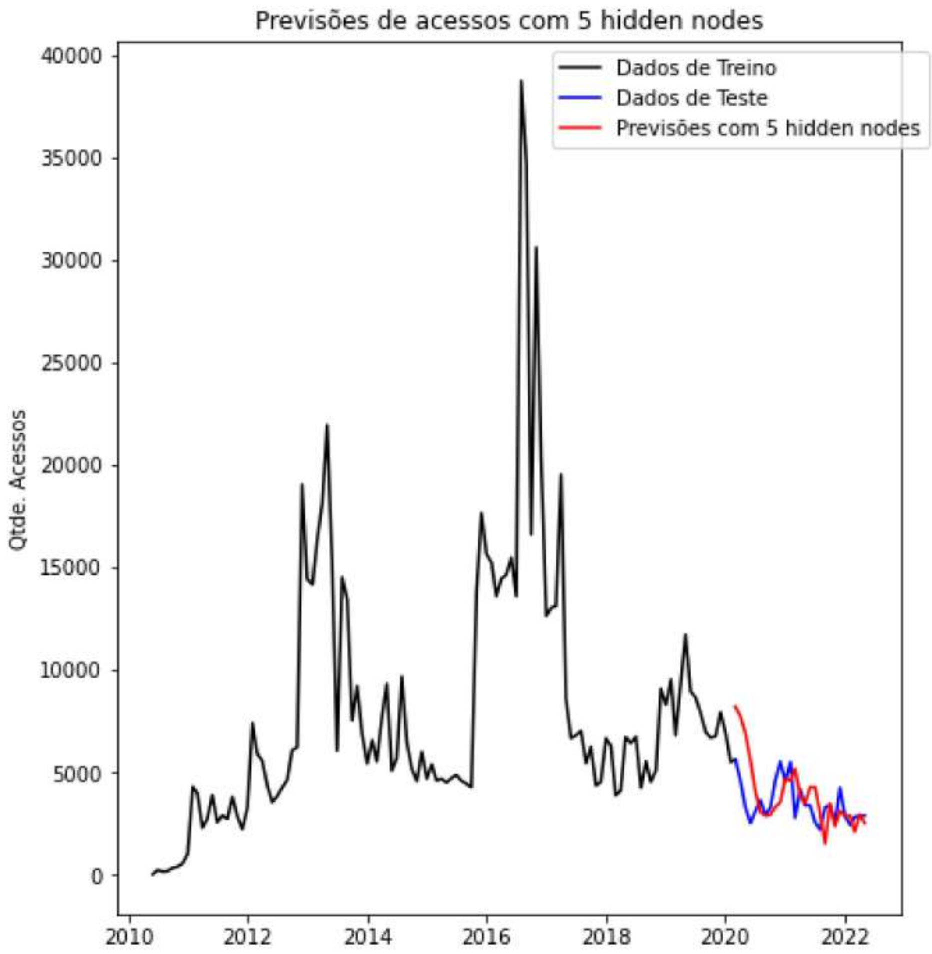
Treinando e testando o modelo com 5 hidden nodes

In [21]:

```
mlp = MLPRegressor(hidden_layer_sizes=(5,), max_iter=500, random_state=1, shuffle=False)

mlp.fit(X_train, y_train)

y_pred = mlp.predict(X_test)
```

In [22]:

```
y_test_ = scaler.inverse_transform(np.hstack([X_test, y_test.reshape(-1, 1)]))[:, -1]
y_pred_ = scaler.inverse_transform(np.hstack([X_test, y_pred.reshape(-1, 1)]))[:, -1]

fig, ax = plt.subplots()
plt.ylabel('Qtde. Acessos')
plt.title('Previsões de acessos com 5 hidden nodes')
plt.plot(VC.iloc[:(k + len(y_train) + 1), :], color='black', label='Dados de Treino')
plt.plot(pd.Series(y_test_, index=VC.index[-len(y_test):]), color='blue', label='Dados de Teste')
plt.plot(pd.Series(y_pred_, index=VC.index[-len(y_test):]), color='red', label='Previsões com 5 hidden nodes')
plt.legend(bbox_to_anchor=(1.05, 1))
ax.xaxis.set_major_formatter(DateFormatter('%Y'))
```



Comentário: *O MLP treinado com 5 hidden nodes apresentou um resultado excelente para essa série temporal, com ajustes quase perfeitos aos dados.*

Treinamento usando Validação Cruzada com 5-Fold

In [23]:

```
tscv = TimeSeriesSplit(n_splits=5)

def cross_validation_loss(hidden_neurons):

    mse = []

    for train_split_, valid_split_ in tscv.split(X_train):
```

```
        X_train_, y_train_ = X_train[train_split_], y_train[train_split_]
        X_valid_, y_valid_ = X_train[valid_split_], y_train[valid_split_]

        mlp = MLPRegressor(hidden_layer_sizes=(hidden_neurons,), max_iter=500, random_state=1, shuffle=False)

        mlp.fit(X_train_, y_train_)

        mse.append(mean_squared_error(y_valid_, mlp.predict(X_valid_)))

    return np.mean(mse)
```

In [24]:

```
params = [5, 10, 15, 25, 50, 75, 100]
mse = [cross_validation_loss(p) for p in params]

params[np.argmin(mse)]
```

Out[24]:

```
10
```

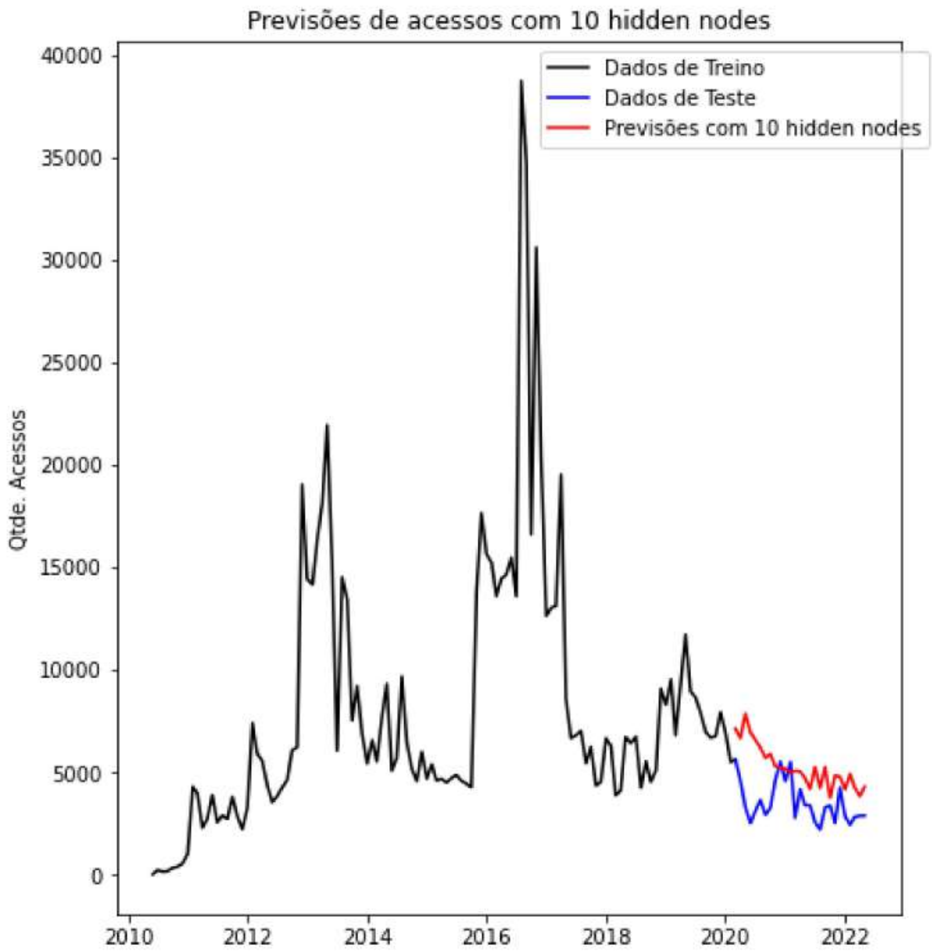
Treinando e testando o modelo com validação cruzada = 5 e hidden nodes = 10

In [25]:

```
mlp = MLPRegressor(hidden_layer_sizes=(10,), max_iter=500, random_state=1, shuffle=False)

mlp.fit(X_train, y_train)

y_pred = mlp.predict(X_test)
```

In [26]:

```
y_test_ = scaler.inverse_transform(np.hstack([X_test, y_test.reshape(-1, 1)]))[:, -1]
y_pred_ = scaler.inverse_transform(np.hstack([X_test, y_pred.reshape(-1, 1)]))[:, -1]

fig, ax = plt.subplots()
plt.ylabel('Qtde. Acessos')
plt.title('Previsões de acessos com 10 hidden nodes')
plt.plot(VC.iloc[:(k + len(y_train) + 1), :], color='black', label='Dados de Treino')
plt.plot(pd.Series(y_test_, index=VC.index[-len(y_test):]), color='blue', label='Dados de Teste')
plt.plot(pd.Series(y_pred_, index=VC.index[-len(y_test):]), color='red', label='Previsões com 10 hidden nodes')
plt.legend(bbox_to_anchor=(1.05, 1))
ax.xaxis.set_major_formatter(DateFormatter('%Y'))
```



Comentário: *O modelo com validação cruzada e 10 hidden nodes também apresentou um bom resultado. Mesmo com menos ajuste aos dados do que o de 5 hidden nodes, é possível que esse seja mais generalizável para novos dados do que os modelos anteriores (MLP e ARIMA).*

Conclusões:

Tanto o modelo criado com ARIMA como os dois criados com o Multilayer Perceptron mostraram que podem contemplar a complexidade dos dados dessa série temporal e fazer boas previsões com eles. Ou seja, qualquer um deles poderia ser usado pela gestão do blog Viagem de Cinema para prever dados futuros, dependendo do objetivo de negócio a ser alcançado.

No entanto, algumas observações precisam ser consideradas em relação às duas técnicas:

- *No caso do ARIMA, como o seu invervalo de confiança abrangeu quase toda a irregularidade dos dados, isso levou a valores extremos, o que dificultaria escolher uma previsão adequada sem contemplar outras decisões de negócios, como campanhas de marketing para o período, por exemplo.*
- *No caso dos dois modelos feitos com o Multilayer Perceptron, ambos se ajustaram muito melhor aos dados dessa série temporal do que o do ARIMA, especialmente o MLP com 5 hidden nodes, cuja previsão foi excelente. O ponto a observar seria saber se ele sofreu ou não de overfitting e se é um modelo que pode ser generalizado para dados novos.*

De qualquer forma, ambas técnicas mostraram que, com os ajustes certos em seus hiperparâmetros, elas podem ser efetivas na modelagem de dados complexos em termos de tendência, sazonalidade e ruídos.