Resolução Exercício Prático

# Aula	43
☑ Ready	☐
☑ Finished	☐
≡ Ciclos	Ciclo 04: Premissas

Objetivo da Aula:

- ☐ Fluxo de trabalho
- ☐ Aplicação completa do ARIMA

Conteúdo:

▼ 1. Fluxo de trabalho

▼ Passo 1: Verificar a Estacionariedade da Série

- Use testes estatísticos como ADF (Augmented Dickey-Fuller)
  - Se a série for estacionária → Vá para o passo 2 e calcule o ACF e PACF.
  - Se a série não for estacionária → Aplica uma diferenciação antes do cálculo do ACF e PACF.

▼ Passo 2: Aplicar PACF e ACF

Com a série estacionária,

- Use o gráfico PACF para identificar o parâmetro (p).
- Use o gráfico ACF para identificar o parâmetro (q).

▼ Passo 3: Fine-Tuning dos parâmetros do ARIMA

- Use a diferenciação manual para determinar o parâmetro (d), como por exemplo, d=1 ou d=2
- Defina o valor de p com base no gráfico ACF aplicado sobre a série diferenciada.
- Defina o valor de q com base no gráfico ACF aplicado sobre a série diferenciada.
- Crie um conjunto de valores alternativos de p e q para serem testados.
- Use a Técnica Rolling Forecast Origin, ajustando os parâmetros do ARIMA com os dados de treinamento e avaliando o desempenho sobre os dados de validação, utilizando métricas de erro como MAPE, AIC e BIC.
- Determinar os melhores parâmetros (p, d, q)

▼ 2. Aplicação completa do ARIMA

▼ Passo 1: Gerando a Série

```
import numpy as np
import seaborn as sns
import pandas as pd
import warnings
from statsmodels.tsa.ar_model import AutoReg as AR
from statsmodels.tsa.arima.model import ARIMA
from matplotlib import pyplot as plt
from statsmodels.graphics.tsaplots import plot_pacf, plot_acf
from statsmodels.tsa.stattools import adfuller


np.random.seed(42)
n = 500

# criacao da serie perfeita
trend = np.linspace( 0, 0, n)
noise = np.random.normal( 0, 1, n)
serie_perfeita = trend + noise
```

```
dates = pd.date_range( start='2023-01-01', periods=n, freq='D' )
serie_perfeita = pd.Series( serie_perfeita, index=dates, name='serie_pe

# quebra premissa 1: linearidade
trend_break = np.linspace( 0, 10, n )
serie_nao_linear = serie_perfeita + trend_break

# quebra premissa 2: estacionariedade
seasonality = 3 * np.sin( 2 * np.pi * np.arange( n ) / 50  )
serie_nao_estacionaria = serie_perfeita + seasonality

# quebra premissa 3: autocorrelacao
autoregressive = np.zeros( n)
autoregressive[0] = noise[0]
for t in range( 1, n ):
    autoregressive[t] = 0.8 * autoregressive[t-1] + np.random.normal( 0, (

serie_nao_autocorrelacao = serie_perfeita + autoregressive

# quebra premissa 4: homocedasticidade
non_normal = noise * np.linspace( 1, 3, n )
serie_nao_homocedastica = serie_perfeita + non_normal

# quebra premissa 5: nao normalidade
non_normal_noise = np.random.exponential( 1, n )
serie_nao_normal = serie_perfeita + non_normal_noise

# combinar as series
serie_final = (
    serie_perfeita
    + trend_break
    + seasonality
    + autoregressive
    + non_normal
    + non_normal_noise
)

serie_final = pd.Series( serie_final, index=dates, name='serie_final' )
```

```
serie_final = serie_final - serie_final.min() + 1

# visualizacao das series
plt.figure( figsize=(16, 8) )
sns.lineplot( serie_final )
```

▼ Passo 2: Separação Train-Test-Validação

```
train_size = int( 0.8 * len( serie_final ) )
validation_size = int( 0.1 * len( serie_final ) )

# split
train = serie_final[:train_size]
validation = serie_final[train_size:train_size + validation_size]
test = serie_final[train_size + validation_size:]
```

▼ Passo 3: Teste Estacionariedade

```
# test adfuller
ad_test = adfuller( train )

# Exibir os resultados do teste
print("Resultado do Teste de Dickey-Fuller:")
print(f"Estatística ADF: {adf_test[0]:.4f}")
print(f"Valor-p: {adf_test[1]:.4f}")

# Interpretação do teste
if adf_test[1] > 0.05:
    print("\nA série NÃO é estacionária (não rejeitamos H0).")
else:
    print("\nA série é estacionária (rejeitamos H0).")
```

▼ Passo 4: Transformando a série em Estacionária

```
serie_diferenciada = train.diff().dropna()
adf_test = adfuller( serie_diferenciada )
```

```
# Exibir os resultados do teste
print("Resultado do Teste de Dickey-Fuller:")
print(f"Estatística ADF: {adf_test[0]:.4f}")
print(f"Valor-p: {adf_test[1]:.4f}")

# Interpretacao do teste
if adf_test[1] > 0.05:
    print( "\nA série NÃO é estacionária (não rejeitamos H0).")

else:
    print( "\nA série é estacionária (rejeitamos H0).")
```

▼ Passo 5: Teste PACF e ACF

```
# teste PACF
#plt.figure( figsize=(12, 6) )
#plot_pacf( serie_diferenciada, lags=20 );

# teste ACF
#plot_acf( serie_diferenciada, lags=20 );

# parametros
p, d, q = 6, 1, 2
```

▼ Passo 6: Modelo ARIMA Fine Tuning

```
# Fine Tuning
p_values = [5, 6, 7]
q_values = [2, 3]
d_values = [1]
num_iterations = 2

results_val = pd.DataFrame()
for _ in range( num_iterations ):
    p = int( np.random.choice( p_values ) )
    q = int( np.random.choice( q_values ) )
    d = int( np.random.choice( d_values ) )
```

```
print( f'Testando parâmetro: p, d, q={p, d, q}' )

predictions = []
actuals = []
train_series = train.copy()
for t in range( len( validation ) ):
    # model definition
    model = ARIMA( train_series, order=(p,d,q) )

    # model training
    model_fit = model.fit()

    # forecast
    forecast = model_fit.forecast( steps=1 ).iloc[0]

    # store predictions
    predictions.append( forecast)
    actuals.append( validation.iloc[t])

    # update training
    train_series = pd.concat( [train_series, pd.Series(validation.iloc[t], i

# prediction x actuals
df_val = pd.DataFrame( {"Predictions": predictions, "Actuals": actuals

# compute metrics
erros = df['Actuals'] - df['Predictions']

rmse_val = np.sqrt( np.mean( errors**2 ) )
mae_val = np.mean( np.abs( errors ) )
mape_val = np.mean( np.abs( errors / df['Actuals'] ) )

log_likelihood = model_fit.llf
num_obs = len( train_series )
num_params = model_fit.params.shape[0]
aic_val = -2 * log_likehood + 2 * num_params
bic_val = -2 * log_likehood + num_params * np.log( num_obs )
```

```
    # store results
    performance = pd.DataFrame( {"P": p, "D": d, "Q": q,
                                 "RMSE_VAL": rmse_val ,
                                 "MAE_VAL": mae_val,
                                 "MAPE_VAL": mape_val,
                                 "AIC_VAL": aic_val,
                                 "BIC_VAL": bic_val } , index=[0] )

    results_val = pd.concat( [results_val, performance] )

results_val.sort_values( "RMSE_VAL", ascending=True )
```

▼ Passo 7: Previsão

```
# previsao do test
p, d, q = 5, 1, 3

predictions = []
actuals = []
results_test = pd.DataFrame()
train_series = pd.concat( [train, validation] )

for t in range( len( test ) ):
    # model definition
    model = ARIMA( train_series, order=(p,d,q) )

    # model training
    model_fit = model.fit()

    # forecast
    forecast = model_fit.forecast( steps=1 ).iloc[0]

    # store predictions
    predictions.append( forecast )
    actuals.append( test.iloc[t])

    # update training
```

```
    train_series = pd.concat( [train_series, pd.Series(test.iloc[t], index=[t

# prediction x actuals
df_test = pd.DataFrame( {"Predictions": predictions, "Actuals": actuals }

# compute metrics
erros = df['Actuals'] - df['Predictions']

rmse_test = np.sqrt( np.mean( errors**2 ) )
mae_test = np.mean( np.abs( errors ) )
mape_test = np.mean( np.abs( errors / df['Actuals'] ) )

log_likelihood = model_fit.llf
num_obs = len( train_series )
num_params = model_fit.params.shape[0]
aic_test = -2 * log_likehood + 2 * num_params
bic_test = -2 * log_likehood + num_params * np.log( num_obs )

performance = pd.DataFrame( {"Q": q, "P": p, "D": d,
                             "RMSE_TEST": rmse_test ,
                             "MAE_TEST": mae_test,
                             "MAPE_TEST": mape_test,
                             "AIC_TEST": aic_test,
                             "BIC_TEST": bic_test } , index=[0] )

results_test = pd.concat( [results_test, performance] )

results_test
```

▼ Passo 8: Visualização dos dados

```
# Visualizacao
plt.figure( figsize=( 16, 8 ) )
sns.lineplot( train, color='black')
sns.lineplot( validation, color='blue')
sns.lineplot( test, color='green')
```

```
sns.lineplot( df_test['Predictions'], color='red')
sns.lineplot( df_test['Actuals'], color='orange')
```