Previsão com os dados de teste

# Aula	18
☑ Ready	☑
☑ Finished	☑
≡ Ciclos	Ciclo 02: Fundamentos

Objetivo da Aula:

- [ ] Previsão sobre os dados de teste

Conteúdo:

▼ 1. Previsão sobre os dados de teste

Após a definição dos melhores parâmetros para o algoritmo, é preciso unir os dados de treinamento e validação, retreinar o modelo com os dados e aplicá-los sobre os dados de teste para medir sua capacidade de generalização (aprendizado) para dados que ele nunca viu.

▼ 1.1. Código em Python para os dados de teste

```
# Best param
best_lag = 3
lag = best_lag

predictions = []
actuals = []
results_test = pd.DataFrame()
```

```
train_series = pd.concat( [train, validation] )
for t in range( len( test ) ):
    # treinamento do modelo
    model = AR( train_series, lags=lag )
    model_fit = model.fit()

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

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

    # update training dataset
    train_series = pd.concat( [train_series, pd.Series( test.iloc[t], index=[tes

df_test = pd.DataFrame( {"Predictions": predictions, "Actuals": actuals}, in

# compute metrics
errors = df_test["Predictions"] - df_test["Actuals"]
rmse_test = np.sqrt( np.mean( errors ** 2 ) )
mae_test = np.mean( np.abs( errors ) )
mape_test = np.mean( np.abs( errors / df_val["Actuals"] ) ) * 100

# calculo das métricas AIC e BIC
log_likelihood = model_fit.llf
num_params = model_fit.params.shape[0]
num_obs = len( train_series )

aic_test = -2 * log_likelihood + 2 * num_params
bic_test = -2 * log_likelihood + num_params * np.log( num_obs )


performance = pd.DataFrame({'Lag': best_lag,
                            "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
```

▼ 1.2. Código Completo até o momento

```
import numpy as np
import seaborn as sns
import pandas as pd
import warnings

from matplotlib import pyplot as plt
from statsmodels.tsa.ar_model import AutoReg as AR

# configuracao da series
np.random.seed(42)
n = 500

# criacao da series 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_perfe

# 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 dos residucos
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, 0.5

serie_nao_autocorrelacao = serie_perfeita + autoregressive

# Quebra Premissa 4: Homoscedasticidade
non_normal = noise * np.linspace( 1, 3, n )
serie_nao_homoscedastica = serie_perfeita + non_normal

# Quebra Premissa 5: Não nomralidade dos residuos
non_normal_noise = np.random.exponential( scale=1, size= n )
serie_nao_normal = trend + 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 )
```

```
# Divisão da Série em Conjuntos de Treinamento, Validação e Teste
train_size = int( 0.8 * len( serie_final ) )
validation_size = int( 0.1 * len( serie_final ) )

train = serie_final[:train_size]
validation = serie_final[train_size:train_size + validation_size]
test = serie_final[train_size + validation_size:]

warnings.filterwarnings( "ignore" )

predictions = []
actuals = []
train_series = train.copy()
for t in range( len( validation ) ):
    # Training Model
    model = AR( train_series, lags=4 )
    model_fit = model.fit()

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

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

    # Update training dataset
    train_series = pd.concat( [train_series, pd.Series( validation.iloc[t], inde

df = pd.DataFrame( {"Predictions": predictions, "Actuals": actuals}, index=

# Best param
best_lag = 3
lag = best_lag

predictions = []
actuals = []
```

```
results_test = pd.DataFrame()
train_series = pd.concat( [train, validation] )
for t in range( len( test ) ):
    # treinamento do modelo
    model = AR( train_series, lags=lag )
    model_fit = model.fit()

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

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

    # update training dataset
    train_series = pd.concat( [train_series, pd.Series( test.iloc[t], index=[tes

df_test = pd.DataFrame( {"Predictions": predictions, "Actuals": actuals}, in

# compute metrics
errors = df_test["Predictions"] - df_test["Actuals"]
rmse_test = np.sqrt( np.mean( errors ** 2 ) )
mae_test = np.mean( np.abs( errors ) )
mape_test = np.mean( np.abs( errors / df_val["Actuals"] ) ) * 100

# calculo das métricas AIC e BIC
log_likelihood = model_fit.llf
num_params = model_fit.params.shape[0]
num_obs = len( train_series )

aic_test = -2 * log_likelihood + 2 * num_params
bic_test = -2 * log_likelihood + num_params * np.log( num_obs )


performance = pd.DataFrame({'Lag': best_lag,
                            "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] )

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

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