##################################################

# Statistical Methods

##################################################

import itertools

import warnings

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

import statsmodels.api as sm

from sklearn.metrics import mean\_absolute\_error

# from statsmodels.tsa.holtwinters import ExponentialSmoothing

# from statsmodels.tsa.holtwinters import SimpleExpSmoothing

from statsmodels.tsa.seasonal import seasonal\_decompose

import statsmodels.tsa.api as smt

from statsmodels.tsa.statespace.sarimax import SARIMAX

warnings.filterwarnings('ignore')

############################

# Veri Seti

############################

# Atmospheric CO2 from Continuous Air Samples at Mauna Loa Observatory, Hawaii, U.S.A.

# Period of Record: March 1958 - December 2001

data = sm.datasets.co2.load\_pandas()

y = data.data

y = y['co2'].resample('MS').mean()

y = y.fillna(y.bfill())

train = y[:'1997-12-01']

test = y['1998-01-01':]

##################################################

# ARIMA(p, d, q): (Autoregressive Integrated Moving Average)

##################################################

model = sm.tsa.arima.ARIMA(train, order=(1, 1, 1))

arima\_model = model.fit()

arima\_model.summary()

y\_pred = arima\_model.get\_forecast(steps=48).predicted\_mean

y\_pred = pd.Series(y\_pred, index=test.index)

def plot\_co2(train, test, y\_pred, title):

    mae = mean\_absolute\_error(test, y\_pred)

    train["1985":].plot(legend=True, label="TRAIN", title=f"{title}, MAE: {round(mae,2)}")

    test.plot(legend=True, label="TEST", figsize=(6, 4))

    y\_pred.plot(legend=True, label="PREDICTION")

    plt.show()

plot\_co2(train, test, y\_pred, "ARIMA")

############################

# Hyperparameter Optimization (Model Derecelerini Belirleme)

############################

############################

# AIC & BIC İstatistiklerine Göre Model Derecesini Belirleme

############################

p = d = q = range(0, 4)

pdq = list(itertools.product(p, d, q))

def arima\_optimizer\_aic(train, orders):

    best\_aic, best\_params = float("inf"), None

    for order in orders:

        try:

            arima\_model = sm.tsa.ARIMA(train, order=order).fit()

            aic = arima\_model.aic

            if aic < best\_aic:

                best\_aic, best\_params = aic, order

            print('ARIMA%s AIC=%.2f' % (order, aic))

        except:

            continue

    print('Best ARIMA%s AIC=%.2f' % (best\_params, best\_aic))

    return best\_params

best\_params\_aic = arima\_optimizer\_aic(train, pdq)

############################

# Final Model

############################

# Fit the final ARIMA model using the best hyperparameters selected by AIC

arima\_model = sm.tsa.ARIMA(train, order=best\_params\_aic).fit()

# Make predictions for the test set

y\_pred = arima\_model.forecast(steps=len(test))

# Convert predictions to a pandas Series with the correct index

y\_pred = pd.Series(y\_pred, index=test.index)

# Plot the actual data and the predictions

plt.figure(figsize=(10, 6))

plt.plot(train, label='Training Data')

plt.plot(test, label='Test Data')

plt.plot(y\_pred, label='ARIMA Predictions')

plt.legend()

plt.xlabel('Year')

plt.ylabel('CO2 Levels (ppm)')

plt.title('ARIMA Model Forecasting')

plt.show()

############################

# Final Model

############################

# Fit the final ARIMA model using the best hyperparameters selected by AIC

arima\_model = sm.tsa.ARIMA(train, order=best\_params\_aic).fit()

# Make predictions for the test set

y\_pred = arima\_model.forecast(steps=len(test))

# Convert predictions to a pandas Series with the correct index

y\_pred = pd.Series(y\_pred, index=test.index)

# Plot the actual data and the predictions

plt.figure(figsize=(10, 6))

plt.plot(train, label='Training Data')

plt.plot(test, label='Test Data')

plt.plot(y\_pred, label='ARIMA Predictions')

plt.legend()

plt.xlabel('Year')

plt.ylabel('CO2 Levels (ppm)')

plt.title('ARIMA Model Forecasting')

plt.show()