```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_squared_error, r2_score

# Load the dataset
df = pd.read_excel('/home/tugce/İndirilenler/DamDataExcel.xlsx')

# Convert the "Date" column to a datetime object and set it as the index
df["Date"] = pd.to_datetime(df["Date"])
df.set_index("Date", inplace=True)
df = df.replace(",", ".", regex=True).replace("%", "", regex=True).astype(float)

# Split the data into training (2017-2019) and testing (2020) sets
train_data = df.loc["2017-01-01":"2019-12-31"]
test_data = df.loc["2020-01-01":"2021-12-31"]

# Initialize a dictionary to store the predictions and evaluation metrics for each dam
results = {}

# Loop through each dam and fit an ARIMA model to the training data, generate predictions for the
test data,
# and evaluate the performance of the model
for dam in train_data.columns:
    # Fit the ARIMA model to the training data
    model = ARIMA(train_data[dam], order=(2, 1, 2))
    fit_model = model.fit()

    start_date = test_data.index[0]
    end_date = test_data.index[-1]
    # Use the new start date in the predict method
    # Get the integer location of the start and end dates in the test data index
    start_loc = test_data.index.get_loc(start_date)
    end_loc = test_data.index.get_loc(end_date)

    # Use the integer locations in the predict method
    predictions = fit_model.predict(start=start_loc, end=end_loc, typ='levels')

    # Evaluate the model using mean absolute percentage error (MAPE)
    def mape(y_true, y_pred):
        return 100 * np.mean(np.abs((y_true - y_pred) / y_true))

    mape_score = mape(test_data[dam], predictions)
    mse_score = mean_squared_error(test_data[dam], predictions)
    r2_score_value = r2_score(test_data[dam], predictions)

    # Add the predictions and evaluation metrics to the results dictionary
    results[dam] = {
        "predictions": predictions,
        "mape": mape_score,
        "mse": mse_score,
```

```
        "r2_score": r2_score_value
    }

    # Plot the actual and predicted values
    plt.figure(figsize=(12, 6))
    plt.plot(train_data[dam], label="Training Data")
    plt.plot(test_data[dam], label="Test Data")
    plt.plot(predictions, label="Predictions")
    plt.title(dam)
    plt.legend(loc="upper left")
    plt.show()

# Print the evaluation metrics for each dam
for dam in results.keys():
    print(f"Results for {dam}:")
    print(f"MAPE score: {results[dam]['mape']}")
    print(f"MSE score: {results[dam]['mse']}")
    print(f"R^2 score: {results[dam]['r2_score']}")
```

My model of choice is the ARIMA model, and it is a good choice for time series forecasting tasks like this one because it is a widely-used and powerful method for modeling and predicting time series data. ARIMA models can capture a wide range of patterns in the data, including trends, seasonal effects, and cycles. The ARIMA model also allows for easy interpretation of the coefficients and underlying statistical assumptions.