arima1

July 8, 2025

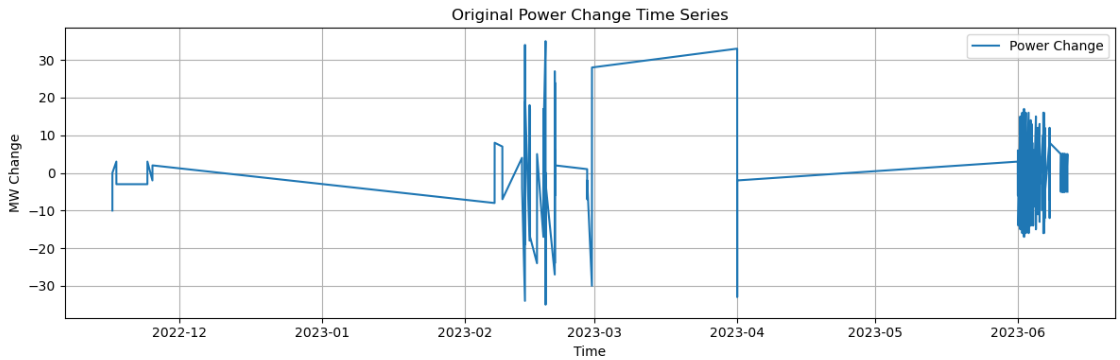
[1]:
```
pip install pmdarima
```

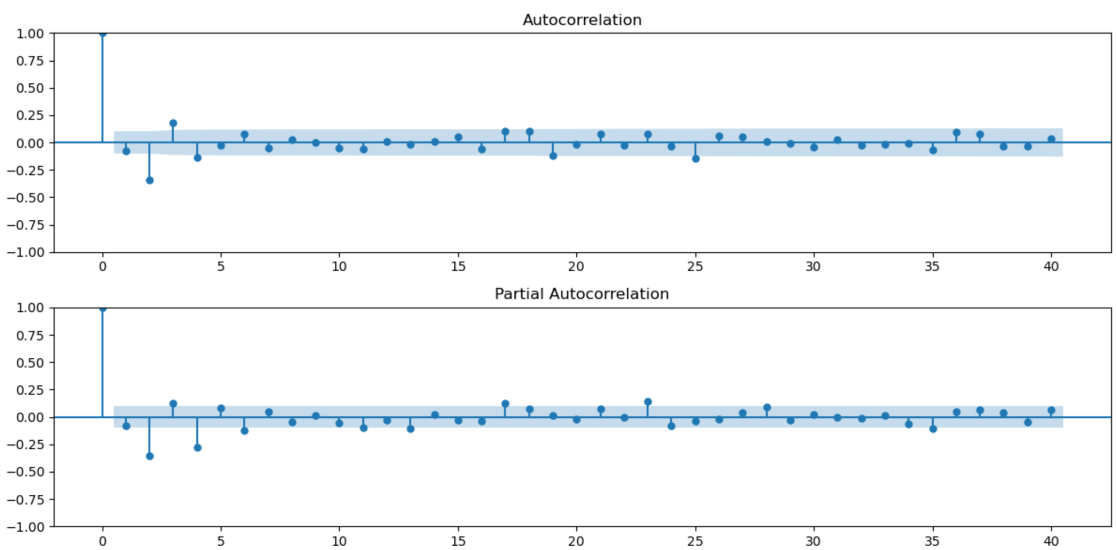
```
Requirement already satisfied: pmdarima in c:\users\bhara\anaconda3\lib\site-
packages (2.0.4)
Requirement already satisfied: joblib>=0.11 in
c:\users\bhara\anaconda3\lib\site-packages (from pmdarima) (1.4.2)
Requirement already satisfied: Cython!=0.29.18,!=0.29.31,>=0.29 in
c:\users\bhara\anaconda3\lib\site-packages (from pmdarima) (3.1.2)
Requirement already satisfied: numpy>=1.21.2 in
c:\users\bhara\anaconda3\lib\site-packages (from pmdarima) (1.26.4)
Requirement already satisfied: pandas>=0.19 in
c:\users\bhara\anaconda3\lib\site-packages (from pmdarima) (2.2.2)
Requirement already satisfied: scikit-learn>=0.22 in
c:\users\bhara\anaconda3\lib\site-packages (from pmdarima) (1.5.1)
Requirement already satisfied: scipy>=1.3.2 in
c:\users\bhara\anaconda3\lib\site-packages (from pmdarima) (1.13.1)
Requirement already satisfied: statsmodels>=0.13.2 in
c:\users\bhara\anaconda3\lib\site-packages (from pmdarima) (0.14.2)
Requirement already satisfied: urllib3 in c:\users\bhara\anaconda3\lib\site-
packages (from pmdarima) (2.2.3)
Requirement already satisfied: setuptools!=50.0.0,>=38.6.0 in
c:\users\bhara\anaconda3\lib\site-packages (from pmdarima) (75.1.0)
Requirement already satisfied: packaging>=17.1 in
c:\users\bhara\anaconda3\lib\site-packages (from pmdarima) (24.1)
Requirement already satisfied: python-dateutil>=2.8.2 in
c:\users\bhara\anaconda3\lib\site-packages (from pandas>=0.19->pmdarima)
(2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in
c:\users\bhara\anaconda3\lib\site-packages (from pandas>=0.19->pmdarima)
(2024.1)
Requirement already satisfied: tzdata>=2022.7 in
c:\users\bhara\anaconda3\lib\site-packages (from pandas>=0.19->pmdarima)
(2023.3)
Requirement already satisfied: threadpoolctl>=3.1.0 in
c:\users\bhara\anaconda3\lib\site-packages (from scikit-learn>=0.22->pmdarima)
(3.5.0)
Requirement already satisfied: patsy>=0.5.6 in
c:\users\bhara\anaconda3\lib\site-packages (from statsmodels>=0.13.2->pmdarima)
```

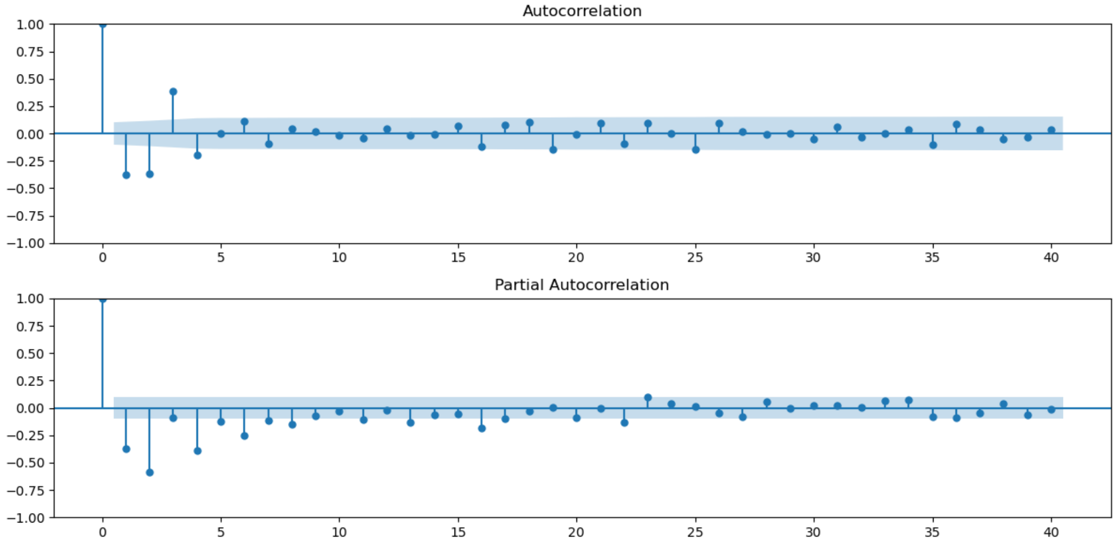
```
(0.5.6)
Requirement already satisfied: six in c:\users\bhara\anaconda3\lib\site-packages
(from patsy>=0.5.6->statsmodels>=0.13.2->pmdarima) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

[3]:
```python
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

# Load dataset
df = pd.read_csv("BOAs.csv")

# Create a new column for power change
df['powerChange'] = df['levelTo'] - df['levelFrom']

# Convert 'timeFrom' to datetime
df['timeFrom'] = pd.to_datetime(df['timeFrom'], utc=True)

# Set datetime as index and sort
df = df.set_index('timeFrom').sort_index()

# Plot original time series
plt.figure(figsize=(12, 4))
plt.plot(df['powerChange'], label='Power Change')
plt.title("Original Power Change Time Series")
plt.xlabel("Time")
plt.ylabel("MW Change")
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()

# ACF & PACF of original series
fig, ax = plt.subplots(2, 1, figsize=(12, 6))
plot_acf(df['powerChange'], ax=ax[0], lags=40)
plot_pacf(df['powerChange'], ax=ax[1], lags=40)
plt.tight_layout()
plt.show()

# Differencing to make stationary
df['powerChange_diff'] = df['powerChange'].diff()

# Drop NA rows caused by differencing
df_cleaned = df.dropna(subset=['powerChange_diff'])

# ACF & PACF of differenced series
fig, ax = plt.subplots(2, 1, figsize=(12, 6))
```

```
plot_acf(df_cleaned['powerChange_diff'], ax=ax[0], lags=40)
plot_pacf(df_cleaned['powerChange_diff'], ax=ax[1], lags=40)
plt.tight_layout()
plt.show()

# Optional: Check basic info of cleaned data
print(df_cleaned[['powerChange', 'powerChange_diff']].info())
```








```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 369 entries, 2022-11-16 11:15:00+00:00 to 2023-06-11
17:05:00+00:00
Data columns (total 2 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   powerChange       369 non-null    int64
 1   powerChange_diff  369 non-null    float64
dtypes: float64(1), int64(1)
memory usage: 8.6 KB
None
```

[7]:

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

# Step 1: Load data
df = pd.read_csv("BOAs.csv")

# Step 2: Compute power change
df['powerChange'] = df['levelTo'] - df['levelFrom']

# Step 3: Convert timeFrom to datetime and set as index
df['timeFrom'] = pd.to_datetime(df['timeFrom'], utc=True)
df = df.set_index('timeFrom').sort_index()
```

```python
# Step 4: Differencing to make stationary
df['powerChange_diff'] = df['powerChange'].diff()

# Step 5: Drop initial NaN from differencing
df_cleaned = df.dropna(subset=['powerChange_diff'])

# Step 6: Use differenced series for modeling
series = df_cleaned['powerChange_diff']

# Step 7: Split into train (85%) and test (15%)
train_size = int(len(series) * 0.85)
train = series[:train_size]
test = series[train_size:]

# Optional: internal validation from train (last 5% of train)
val_size = int(len(train) * 0.05)
train_main = train[:-val_size]
val = train[-val_size:]

# Step 8: Fit ARIMA model manually (choose p,d,q based on ACF/PACF)
model = ARIMA(train_main, order=(2, 0, 2))  # d=0 because already differenced
model_fit = model.fit()

# Step 9: Forecast on validation + test
n_forecast = len(val) + len(test)
forecast = model_fit.forecast(steps=n_forecast)

# Step 10: Forecast on validation + test
n_forecast = len(val) + len(test)
forecast_values = model_fit.forecast(steps=n_forecast)

# Step 11: Reindex both for alignment (drop any NaNs in actual)
actual = pd.concat([val, test]).dropna().reset_index(drop=True)
forecast = pd.Series(forecast_values[:len(actual)], index=actual.index)

#  Debug check
print("Actual length:", len(actual))
print("Forecast length:", len(forecast))

# Step 12: Evaluation
if len(actual) > 0 and len(forecast) > 0:
    rmse = np.sqrt(mean_squared_error(actual, forecast))
    r2 = r2_score(actual, forecast)

    print(f"\n RMSE: {rmse:.3f}")
    print(f" R² Score: {r2:.3f}")
```

```python
    # Plot
    plt.figure(figsize=(12, 4))
    plt.plot(actual, label='Actual')
    plt.plot(forecast, label='Forecast', linestyle='--')
    plt.title(f"ARIMA Forecast vs Actual (RMSE={rmse:.2f}, R²={r2:.2f})")
    plt.xlabel("Time Steps")
    plt.ylabel("Differenced Power Change")
    plt.grid(True)
    plt.legend()
    plt.tight_layout()
    plt.show()

    # Step 13: Forecast next 10 future values
    future_forecast = model_fit.forecast(steps=10)
    print("\n Next 10 Predicted Differenced Power Change Values:")
    print(future_forecast)

else:
    print("Forecast or Actual is empty - evaluation skipped.")
```

```
Actual length: 71
Forecast length: 71
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[7], line 59
     57 # Step 12: Evaluation
     58 if len(actual) > 0 and len(forecast) > 0:
---> 59     rmse = np.sqrt(mean_squared_error(actual, forecast))
     60     r2 = r2_score(actual, forecast)
     62     print(f"\n RMSE: {rmse:.3f}")

File ~\anaconda3\Lib\site-packages\sklearn\utils\_param_validation.py:213, in␣
 ↪validate_params.<locals>.decorator.<locals>.wrapper(*args, **kwargs)
    207 try:
    208     with config_context(
    209         skip_parameter_validation=(
    210             prefer_skip_nested_validation or global_skip_validation
    211         )
    212     ):
--> 213         return func(*args, **kwargs)
    214 except InvalidParameterError as e:
    215     # When the function is just a wrapper around an estimator, we allow
    216     # the function to delegate validation to the estimator, but we␣
 ↪replace
    217     # the name of the estimator by the name of the function in the erro
    218     # message to avoid confusion.
```

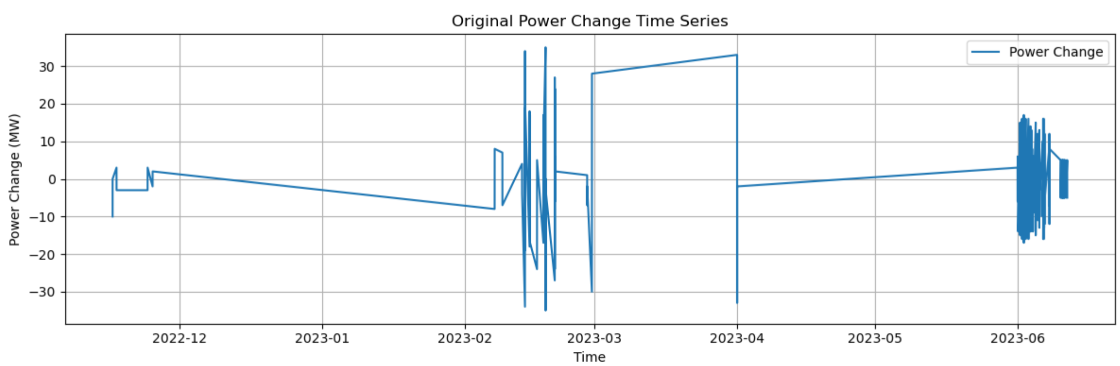
```
    219     msg = re.sub(
    220         r"parameter of \w+ must be",
    221         f"parameter of {func.__qualname__} must be",
    222         str(e),
    223     )

File ~\anaconda3\Lib\site-packages\sklearn\metrics\_regression.py:506, in mean_squared_error(y_true, y_pred, sample_weight, multioutput, squared)
    501     if not squared:
    502         return root_mean_squared_error(
    503             y_true, y_pred, sample_weight=sample_weight, multioutput=multioutput
    504         )
--> 506 y_type, y_true, y_pred, multioutput = _check_reg_targets(
    507     y_true, y_pred, multioutput
    508 )
    509 check_consistent_length(y_true, y_pred, sample_weight)
    510 output_errors = np.average((y_true - y_pred) ** 2, axis=0, weights=sample_weight)

File ~\anaconda3\Lib\site-packages\sklearn\metrics\_regression.py:113, in _check_reg_targets(y_true, y_pred, multioutput, dtype, xp)
    111 check_consistent_length(y_true, y_pred)
    112 y_true = check_array(y_true, ensure_2d=False, dtype=dtype)
--> 113 y_pred = check_array(y_pred, ensure_2d=False, dtype=dtype)
    115 if y_true.ndim == 1:
    116     y_true = xp.reshape(y_true, (-1, 1))

File ~\anaconda3\Lib\site-packages\sklearn\utils\validation.py:1064, in check_array(array, accept_sparse, accept_large_sparse, dtype, order, copy, force_writeable, force_all_finite, ensure_2d, allow_nd, ensure_min_samples, ensure_min_features, estimator, input_name)
   1058     raise ValueError(
   1059         "Found array with dim %d. %s expected <= 2."
   1060         % (array.ndim, estimator_name)
   1061     )
   1063 if force_all_finite:
-> 1064     _assert_all_finite(
   1065         array,
   1066         input_name=input_name,
   1067         estimator_name=estimator_name,
   1068         allow_nan=force_all_finite == "allow-nan",
   1069     )
   1071 if copy:
   1072     if _is_numpy_namespace(xp):
   1073         # only make a copy if `array` and `array_orig` may share memory
```

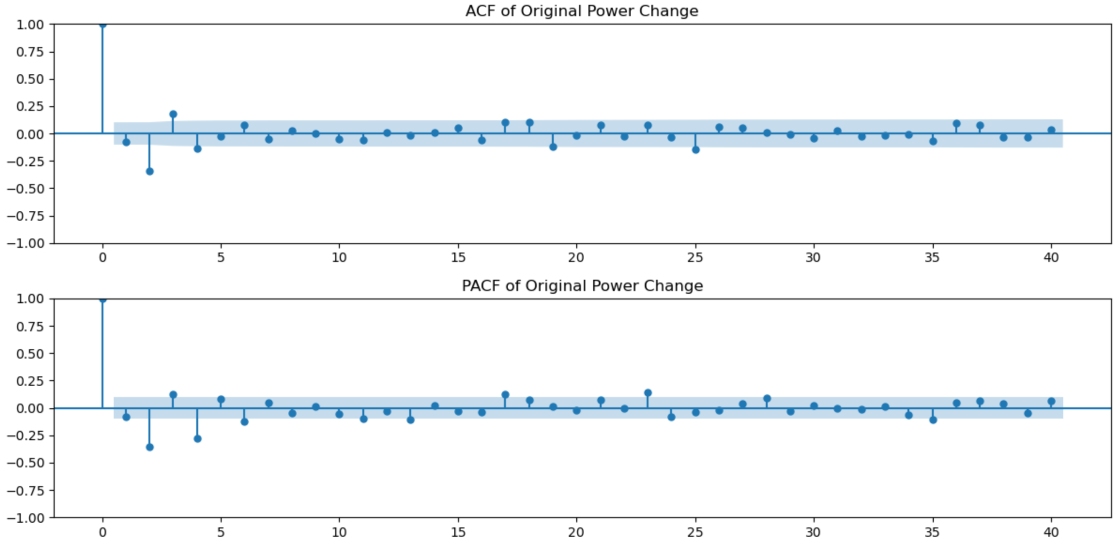
```
File ~\anaconda3\Lib\site-packages\sklearn\utils\validation.py:123, in␣
↪_assert_all_finite(X, allow_nan, msg_dtype, estimator_name, input_name)
    120 if first_pass_isfinite:
    121     return
--> 123 _assert_all_finite_element_wise(
    124     X,
    125     xp=xp,
    126     allow_nan=allow_nan,
    127     msg_dtype=msg_dtype,
    128     estimator_name=estimator_name,
    129     input_name=input_name,
    130 )

File ~\anaconda3\Lib\site-packages\sklearn\utils\validation.py:172, in␣
↪_assert_all_finite_element_wise(X, xp, allow_nan, msg_dtype, estimator_name,␣
↪input_name)
    155 if estimator_name and input_name == "X" and has_nan_error:
    156     # Improve the error message on how to handle missing values in
    157     # scikit-learn.
    158     msg_err += (
    159         f"\n{estimator_name} does not accept missing values"
    160         " encoded as NaN natively. For supervised learning, you might␣
↪want"
   (…)
    170         "#estimators-that-handle-nan-values"
    171     )
--> 172 raise ValueError(msg_err)

ValueError: Input contains NaN.
```

[8]:

```python
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

# Step 1: Load and process data
df = pd.read_csv("BOAs.csv")

# Step 2: Calculate power change
df['powerChange'] = df['levelTo'] - df['levelFrom']

# Step 3: Parse 'timeFrom' as datetime and set index
df['timeFrom'] = pd.to_datetime(df['timeFrom'], utc=True)
df = df.set_index('timeFrom').sort_index()

# Step 4: Drop rows with NaN in powerChange (if any)
df = df.dropna(subset=['powerChange'])
```

```
# Step 5: Plot original powerChange time series
plt.figure(figsize=(12, 4))
plt.plot(df['powerChange'], label='Power Change')
plt.title("Original Power Change Time Series")
plt.xlabel("Time")
plt.ylabel("Power Change (MW)")
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()

# Step 6: Plot ACF and PACF
fig, ax = plt.subplots(2, 1, figsize=(12, 6))

# ACF plot
plot_acf(df['powerChange'], ax=ax[0], lags=40)
ax[0].set_title("ACF of Original Power Change")

# PACF plot
plot_pacf(df['powerChange'], ax=ax[1], lags=40)
ax[1].set_title("PACF of Original Power Change")

plt.tight_layout()
plt.show()
```

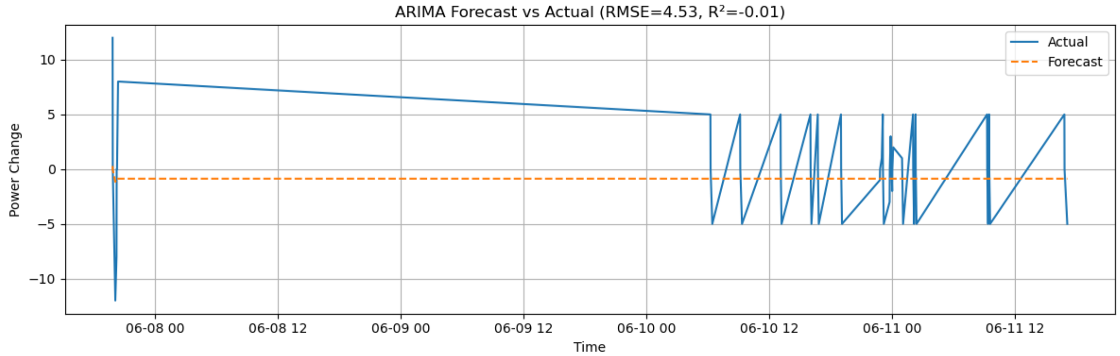




[9]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_squared_error, r2_score
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

# Step 1: Load data
df = pd.read_csv("BOAs.csv")

# Step 2: Compute power change
df['powerChange'] = df['levelTo'] - df['levelFrom']

# Step 3: Parse 'timeFrom' as datetime and set as index
df['timeFrom'] = pd.to_datetime(df['timeFrom'], utc=True)
df = df.set_index('timeFrom').sort_index()

# Step 4: Drop NaNs
df = df.dropna(subset=['powerChange'])

# Step 5: Use powerChange series
series = df['powerChange']

# Step 6: Train-test split
train_size = int(len(series) * 0.85)
train = series[:train_size]
test = series[train_size:]
```

```python
# Step 7: Fit ARIMA model (choose order from ACF/PACF manually; example:␣
↪(2,0,2))
model = ARIMA(train, order=(2, 0, 2))
model_fit = model.fit()

# Step 8: Forecast for test period
forecast_test = model_fit.forecast(steps=len(test))
forecast_test.index = test.index  # align index for plotting

# Step 9: Evaluation
rmse = np.sqrt(mean_squared_error(test, forecast_test))
r2 = r2_score(test, forecast_test)

print(f" Evaluation on Test Set:\nRMSE: {rmse:.3f}\nR² Score: {r2:.3f}")

# Step 10: Plot Actual vs Forecasted (Test)
plt.figure(figsize=(12, 4))
plt.plot(test, label='Actual')
plt.plot(forecast_test, label='Forecast', linestyle='--')
plt.title(f"ARIMA Forecast vs Actual (RMSE={rmse:.2f}, R²={r2:.2f})")
plt.xlabel("Time")
plt.ylabel("Power Change")
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()

# Step 11: Forecast next 10 future time steps (beyond the last date)
future_forecast = model_fit.forecast(steps=10)

# Optional: Assign future timestamps based on previous frequency (if regular)
if df.index.inferred_freq:
    future_index = pd.date_range(start=df.index[-1], periods=11, freq=df.index.
↪inferred_freq)[1:]
else:
    future_index = range(1, 11)  # fallback to generic range

future_forecast = pd.Series(future_forecast, index=future_index)

# Step 12: Print future predictions
print(future_forecast)
```

```
 Evaluation on Test Set:
RMSE: 4.528
R² Score: -0.013
```




```
 Forecasted Power Change for Next 10 Steps:
1     NaN
2     NaN
3     NaN
4     NaN
5     NaN
6     NaN
7     NaN
8     NaN
9     NaN
10    NaN
Name: predicted_mean, dtype: float64
```

[12]:
```
print(future_forecast.describe())
```

```
count    0.0
mean     NaN
std      NaN
min      NaN
25%      NaN
50%      NaN
75%      NaN
max      NaN
Name: predicted_mean, dtype: float64
```

[13]:
```
future_df = future_forecast.to_frame(name="Forecasted Power Change")
future_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 1 to 10
Data columns (total 1 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Forecasted Power Change  0 non-null      float64
```

```
dtypes: float64(1)
memory usage: 212.0 bytes
```

[14]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from pmdarima import auto_arima
from sklearn.metrics import mean_squared_error, r2_score

# Step 1: Load and prepare the dataset
df = pd.read_csv("BOAs.csv")
df['powerChange'] = df['levelTo'] - df['levelFrom']
df['timeFrom'] = pd.to_datetime(df['timeFrom'], utc=True)
df = df.set_index('timeFrom').sort_index()
df = df.dropna(subset=['powerChange'])

series = df['powerChange']

# Step 2: Train-test split
train_size = int(len(series) * 0.85)
train = series[:train_size]
test = series[train_size:]

# Step 3: Apply auto_arima to find best model
stepwise_model = auto_arima(
    train,
    start_p=1, start_q=1,
    max_p=5, max_q=5,
    seasonal=False,
    d=None,            # Let it test for stationarity and pick d
    trace=True,
    error_action='ignore',
    suppress_warnings=True,
    stepwise=True
)

# Step 4: Fit the best model
stepwise_model.fit(train)

# Step 5: Forecast test set
forecast_test = stepwise_model.predict(n_periods=len(test))
forecast_test = pd.Series(forecast_test, index=test.index)

# Step 6: Evaluation
rmse = np.sqrt(mean_squared_error(test, forecast_test))
r2 = r2_score(test, forecast_test)
```

```
print(f"\n  Auto-ARIMA Model Evaluation:")
print(f"Best ARIMA order: {stepwise_model.order}")
print(f"RMSE: {rmse:.3f}")
print(f"R² Score: {r2:.3f}")

# Step 7: Plot actual vs forecast
plt.figure(figsize=(12, 4))
plt.plot(test, label='Actual')
plt.plot(forecast_test, label='Forecast (Auto ARIMA)', linestyle='--')
plt.title(f"Auto ARIMA Forecast vs Actual (RMSE={rmse:.2f}, R²={r2:.2f})")
plt.xlabel("Time")
plt.ylabel("Power Change")
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()

# Step 8: Forecast next 10 future time steps
future_forecast = stepwise_model.predict(n_periods=10)

# Optional: Assign timestamps if possible
if df.index.inferred_freq:
    future_index = pd.date_range(start=df.index[-1], periods=11, freq=df.index.
↪inferred_freq)[1:]
else:
    future_index = range(1, 11)

future_forecast = pd.Series(future_forecast, index=future_index)

print("\n  Forecasted Power Change for Next 10 Steps (Auto ARIMA):")
print(future_forecast)

print("\n  Summary Stats:")
print(future_forecast.describe())
```

```
Performing stepwise search to minimize aic
 ARIMA(1,0,1)(0,0,0)[0]             : AIC=2296.865, Time=0.18 sec
 ARIMA(0,0,0)(0,0,0)[0]             : AIC=2323.116, Time=0.02 sec
 ARIMA(1,0,0)(0,0,0)[0]             : AIC=2323.498, Time=0.02 sec
 ARIMA(0,0,1)(0,0,0)[0]             : AIC=2320.738, Time=0.03 sec
 ARIMA(2,0,1)(0,0,0)[0]             : AIC=2260.307, Time=0.10 sec
 ARIMA(2,0,0)(0,0,0)[0]             : AIC=2286.340, Time=0.06 sec
 ARIMA(3,0,1)(0,0,0)[0]             : AIC=2262.142, Time=0.20 sec
 ARIMA(2,0,2)(0,0,0)[0]             : AIC=2261.444, Time=0.10 sec
 ARIMA(1,0,2)(0,0,0)[0]             : AIC=2260.820, Time=0.08 sec
 ARIMA(3,0,0)(0,0,0)[0]             : AIC=2282.520, Time=0.05 sec
 ARIMA(3,0,2)(0,0,0)[0]             : AIC=2260.898, Time=0.14 sec
 ARIMA(2,0,1)(0,0,0)[0] intercept   : AIC=2257.796, Time=0.13 sec
```

```
 ARIMA(1,0,1)(0,0,0)[0] intercept   : AIC=2301.367, Time=0.09 sec
 ARIMA(2,0,0)(0,0,0)[0] intercept   : AIC=2282.284, Time=0.06 sec
 ARIMA(3,0,1)(0,0,0)[0] intercept   : AIC=2259.365, Time=0.20 sec
 ARIMA(2,0,2)(0,0,0)[0] intercept   : AIC=2256.746, Time=0.22 sec
 ARIMA(1,0,2)(0,0,0)[0] intercept   : AIC=2254.869, Time=0.22 sec
 ARIMA(0,0,2)(0,0,0)[0] intercept   : AIC=2257.330, Time=0.07 sec
 ARIMA(1,0,3)(0,0,0)[0] intercept   : AIC=2256.849, Time=0.15 sec
 ARIMA(0,0,1)(0,0,0)[0] intercept   : AIC=2318.717, Time=0.06 sec
 ARIMA(0,0,3)(0,0,0)[0] intercept   : AIC=2255.337, Time=0.08 sec
 ARIMA(2,0,3)(0,0,0)[0] intercept   : AIC=2251.767, Time=0.25 sec
 ARIMA(3,0,3)(0,0,0)[0] intercept   : AIC=2249.579, Time=0.32 sec
 ARIMA(3,0,2)(0,0,0)[0] intercept   : AIC=2252.582, Time=0.29 sec
 ARIMA(4,0,3)(0,0,0)[0] intercept   : AIC=2250.998, Time=0.35 sec
 ARIMA(3,0,4)(0,0,0)[0] intercept   : AIC=2250.869, Time=0.34 sec
 ARIMA(2,0,4)(0,0,0)[0] intercept   : AIC=2255.182, Time=0.38 sec
 ARIMA(4,0,2)(0,0,0)[0] intercept   : AIC=2253.075, Time=0.42 sec
 ARIMA(4,0,4)(0,0,0)[0] intercept   : AIC=2252.839, Time=0.66 sec
 ARIMA(3,0,3)(0,0,0)[0]             : AIC=2256.945, Time=0.21 sec

Best model:  ARIMA(3,0,3)(0,0,0)[0] intercept
Total fit time: 5.545 seconds
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[14], line 42
     39 forecast_test = pd.Series(forecast_test, index=test.index)
     41 # Step 6: Evaluation
---> 42 rmse = np.sqrt(mean_squared_error(test, forecast_test))
     43 r2 = r2_score(test, forecast_test)
     45 print(f"\n  Auto-ARIMA Model Evaluation:")

File ~\anaconda3\Lib\site-packages\sklearn\utils\_param_validation.py:213, in␣
 ↪validate_params.<locals>.decorator.<locals>.wrapper(*args, **kwargs)
    207 try:
    208     with config_context(
    209         skip_parameter_validation=(
    210             prefer_skip_nested_validation or global_skip_validation
    211         )
    212     ):
--> 213         return func(*args, **kwargs)
    214 except InvalidParameterError as e:
    215     # When the function is just a wrapper around an estimator, we allow
    216     # the function to delegate validation to the estimator, but we␣
 ↪replace
    217     # the name of the estimator by the name of the function in the erro
    218     # message to avoid confusion.
    219     msg = re.sub(
```

```
    220         r"parameter of \w+ must be",
    221         f"parameter of {func.__qualname__} must be",
    222         str(e),
    223     )

File ~\anaconda3\Lib\site-packages\sklearn\metrics\_regression.py:506, in mean_squared_error(y_true, y_pred, sample_weight, multioutput, squared)
    501     if not squared:
    502         return root_mean_squared_error(
    503             y_true, y_pred, sample_weight=sample_weight, multioutput=multioutput
    504         )
--> 506 y_type, y_true, y_pred, multioutput = _check_reg_targets(
    507     y_true, y_pred, multioutput
    508 )
    509 check_consistent_length(y_true, y_pred, sample_weight)
    510 output_errors = np.average((y_true - y_pred) ** 2, axis=0, weights=sample_weight)

File ~\anaconda3\Lib\site-packages\sklearn\metrics\_regression.py:113, in _check_reg_targets(y_true, y_pred, multioutput, dtype, xp)
    111 check_consistent_length(y_true, y_pred)
    112 y_true = check_array(y_true, ensure_2d=False, dtype=dtype)
--> 113 y_pred = check_array(y_pred, ensure_2d=False, dtype=dtype)
    115 if y_true.ndim == 1:
    116     y_true = xp.reshape(y_true, (-1, 1))

File ~\anaconda3\Lib\site-packages\sklearn\utils\validation.py:1064, in check_array(array, accept_sparse, accept_large_sparse, dtype, order, copy, force_writeable, force_all_finite, ensure_2d, allow_nd, ensure_min_samples, ensure_min_features, estimator, input_name)
   1058     raise ValueError(
   1059         "Found array with dim %d. %s expected <= 2."
   1060         % (array.ndim, estimator_name)
   1061     )
   1063 if force_all_finite:
-> 1064     _assert_all_finite(
   1065         array,
   1066         input_name=input_name,
   1067         estimator_name=estimator_name,
   1068         allow_nan=force_all_finite == "allow-nan",
   1069     )
   1071 if copy:
   1072     if _is_numpy_namespace(xp):
   1073         # only make a copy if `array` and `array_orig` may share memory

File ~\anaconda3\Lib\site-packages\sklearn\utils\validation.py:123, in _assert_all_finite(X, allow_nan, msg_dtype, estimator_name, input_name)
```

```
    120 if first_pass_isfinite:
    121     return
--> 123 _assert_all_finite_element_wise(
    124     X,
    125     xp=xp,
    126     allow_nan=allow_nan,
    127     msg_dtype=msg_dtype,
    128     estimator_name=estimator_name,
    129     input_name=input_name,
    130 )

File ~\anaconda3\Lib\site-packages\sklearn\utils\validation.py:172, in␣
 ↪_assert_all_finite_element_wise(X, xp, allow_nan, msg_dtype, estimator_name,␣
 ↪input_name)
    155 if estimator_name and input_name == "X" and has_nan_error:
    156     # Improve the error message on how to handle missing values in
    157     # scikit-learn.
    158     msg_err += (
    159         f"\n{estimator_name} does not accept missing values"
    160         " encoded as NaN natively. For supervised learning, you might␣
 ↪want"
   (…)
    170         "#estimators-that-handle-nan-values"
    171     )
--> 172 raise ValueError(msg_err)

ValueError: Input contains NaN.
```

[ ]: