Time-series: Seasonal Autoregressive Integrated Moving Average (SARIMA)

Supervised parametric machine learning to predict the future values of a time-series.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.stattools import adfuller
from pmdarima import auto_arima
```

Dataset

This data was obtained from the City of Calgary's open data portal and contains the hourly solar output in kWh from a solar photovoltaic site at the *Bearspaw Water Treatment Plant*.

To make the analysis simpler, solar outputs were converted into monthly averages.

In [2]:
```
#load the data
df = pd.read_csv("https://raw.githubusercontent.com/dswede43/ML-methods/main/data/c

#filter the data by the datetime
df = df[df['date'] <= '2023-08-31 0:00']

#create new date columns
df['date'] = pd.to_datetime(df['date'])
df['year'] = df['date'].dt.year
df['month'] = df['date'].dt.month
df['day'] = df['date'].dt.day

#calculate the monthly averages
df = df.groupby([df['year'], df['month']])['kWh'].mean()
df = pd.DataFrame(df)
df.reset_index(inplace = True)
df['date'] = pd.to_datetime(df['year'].astype(str) + '-' + df['month'].astype(str),
df.head()
```

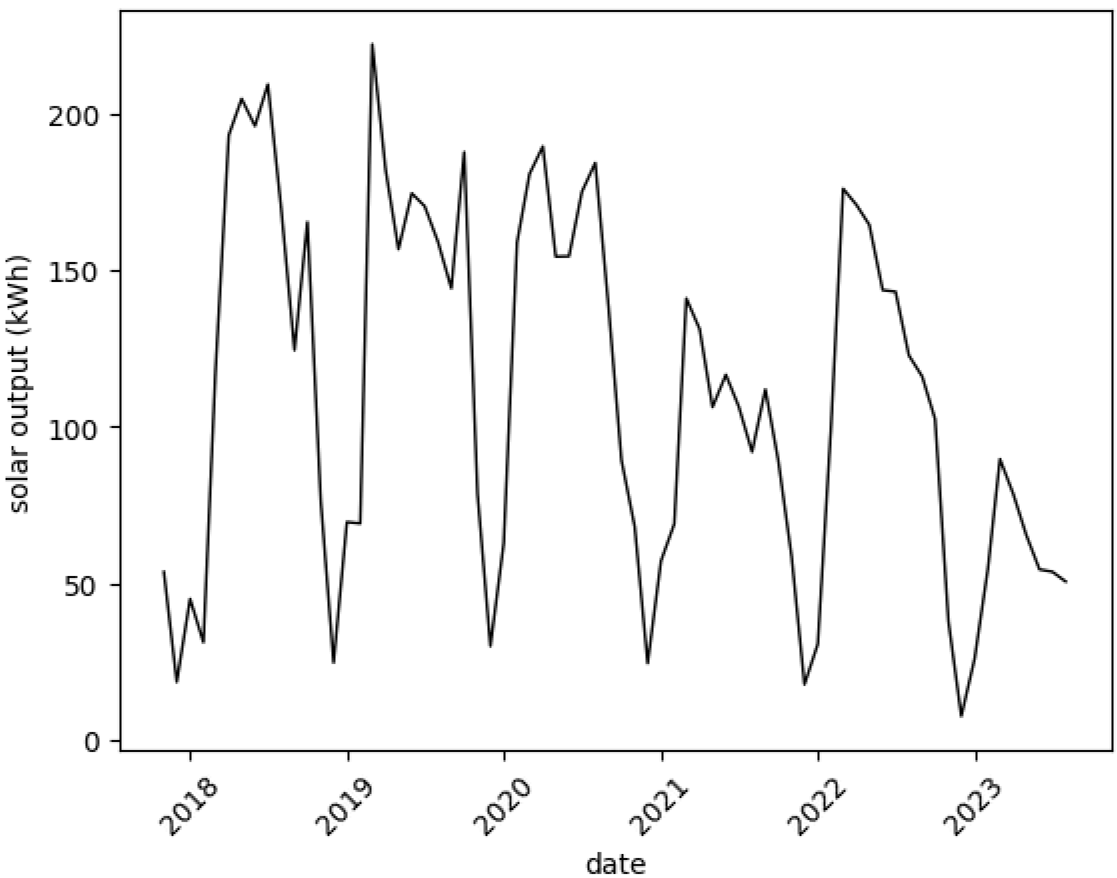
Out[2]:

	year	month	kWh	date
0	2017	11	53.696675	2017-11-01
1	2017	12	18.551453	2017-12-01
2	2018	1	45.098493	2018-01-01
3	2018	2	31.226360	2018-02-01
4	2018	3	117.454937	2018-03-01

Run sequence plots

Visualize the time-series data across time.

In [3]:
```
#plot the run sequence plot
sns.lineplot(data = df, x = 'date', y = 'kWh', linewidth = 1, color = 'black')
plt.xticks(rotation = 45)
plt.xlabel('date')
plt.ylabel('solar output (kWh)')
plt.show()
```



Run sequence plot shows a strong yearly seasonal trend in the data.

Seasonal plots

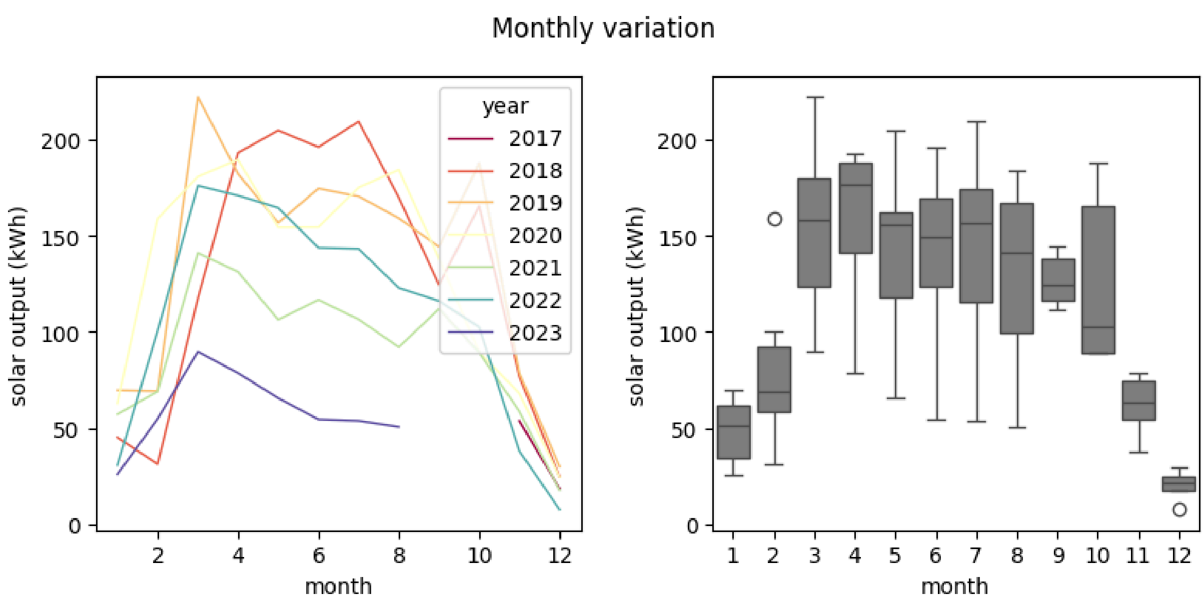
In [4]:
```
#create line and boxplots
fig, axes = plt.subplots(nrows = 1, ncols = 2, figsize = (8, 4))
fig.suptitle("Monthly variation")

#monthly variation lineplot
sns.lineplot(data = df, x = 'month', y = 'kWh',
             linewidth = 1,
             hue = 'year',
             palette = sns.color_palette('Spectral', as_cmap = True),
             ax = axes[0])

#monthly variation boxplot
sns.boxplot(data = df, x = 'month', y = 'kWh', color = 'gray', ax = axes[1])

#set the x and y-axis labels
axes[0].set_ylabel('solar output (kWh)')
axes[1].set_ylabel('solar output (kWh)')

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



The seasonal plots show a seasonal trend in solar output with greater output in the summer months and less in the winter months.

Checking for stationarity

Checking if the mean, variance, and autocorrelation of the data is constant over time.

ACF and PACF plots

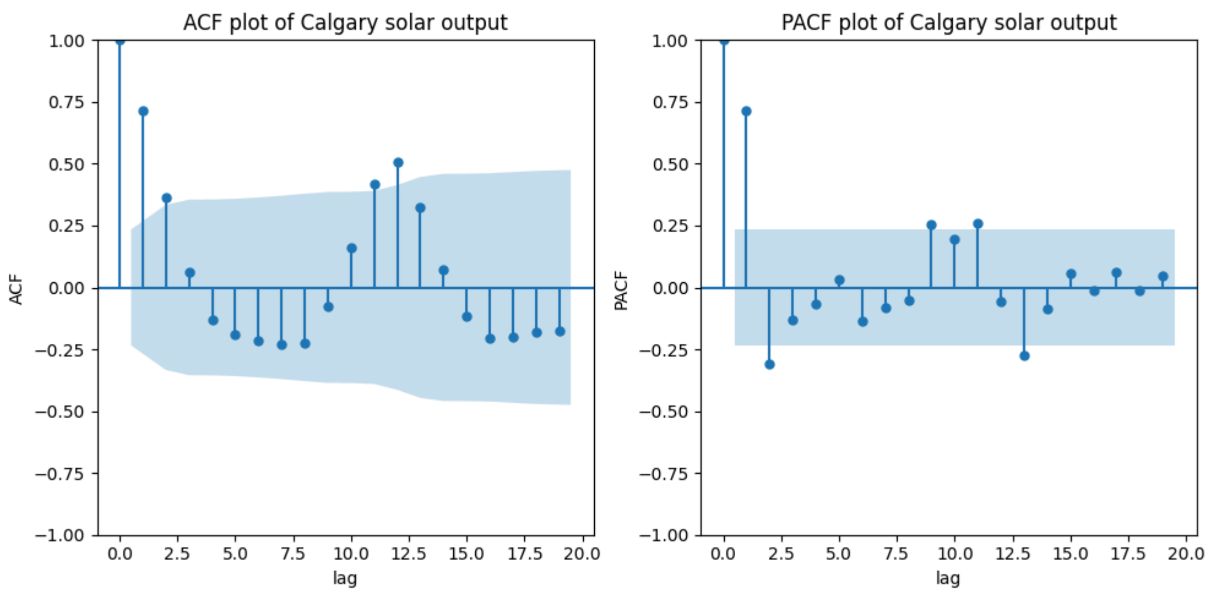
In [5]:
```
#create ACF and PACF plots
fig, axes = plt.subplots(nrows = 1, ncols = 2, figsize = (10, 5))
```

```
#create the ACF plot
plot_acf(df['kWh'], title = 'ACF plot of Calgary solar output', ax = axes[0])

#create the PACF plot
plot_pacf(df['kWh'], title = 'PACF plot of Calgary solar output', ax = axes[1])

#set the axes titles
axes[0].set_xlabel('lag')
axes[0].set_ylabel('ACF')
axes[1].set_xlabel('lag')
axes[1].set_ylabel('PACF')

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



ACF and PACF plots shows autocorrelations present between current and lagged values of itself that are changing over time. Therefor , the data is not independent and exhibits a temporal dependence structure that is likely not stationary.

Dickey-Fuller test for stationarity

$$H_0 : \text{data is non-stationary}$$

$$H_A : \text{data is stationary}$$

In [6]:
```
#complete the Dickey-Fuller test
adfuller_test = adfuller(df['kWh'])
print(f"ADF Statistic: {adfuller_test[0]:.4f}")
print(f"p-value: {adfuller_test[1]:.4f}")
```

```
ADF Statistic: 0.5237
p-value: 0.9856
```

The p-value is >0.05 so the null hypothesis fails to be rejected and confirms the ACF and PACF plots that the data is likely non-stationary.

Differencing

Apply differencing to reduce the effect of seasonality in the data and make it stationary.

1st-order differencing

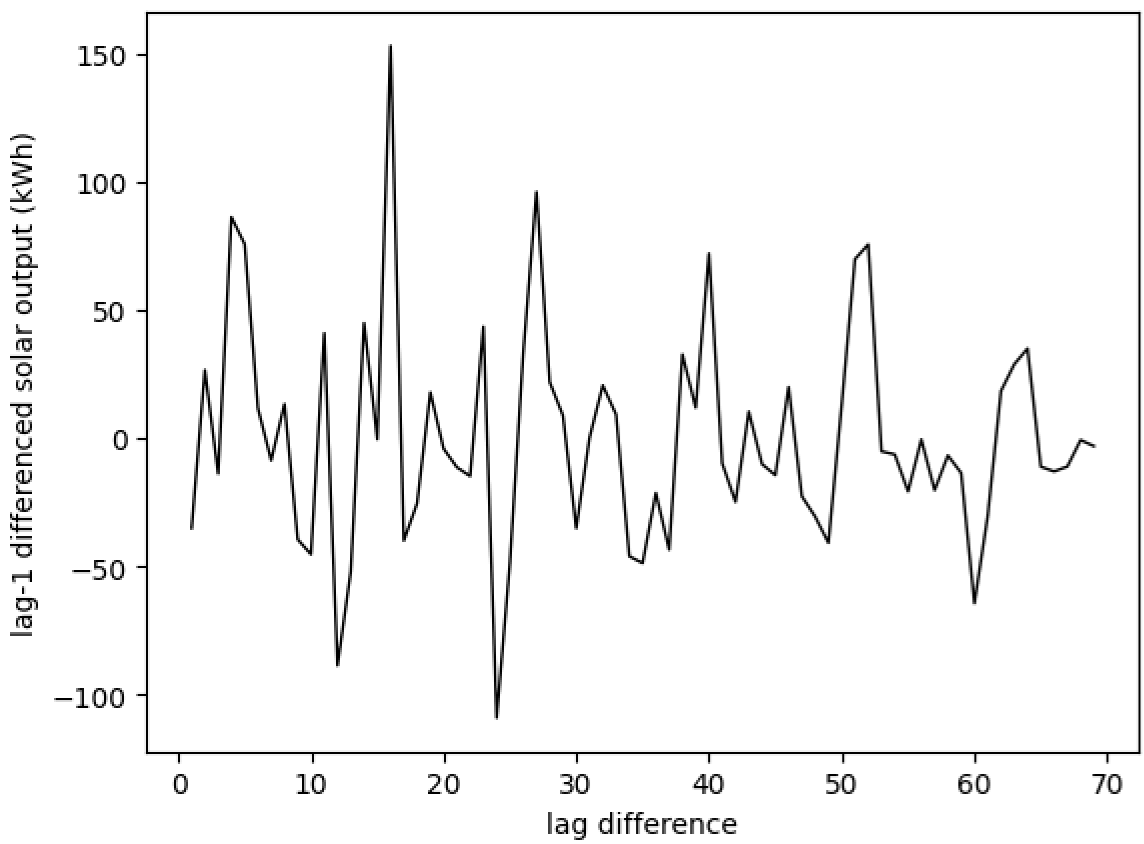
$$y'_t = y_t - y_{t-1}$$

Instead of predicting y_t directly, predict the gap between y_t and y_{t-1}, because we can predict y'_t, we can then reconstruct y_t by

$$y_t = y'_t + y_{t-1}$$

In [7]:
```
#apply 1st-order differencing to the data
df_diff = df['kWh'].diff()
df_diff.dropna(inplace=True)
df_diff = pd.DataFrame(df_diff)

#plot the differenced data
sns.lineplot(data = df_diff, x = df_diff.index, y = 'kWh', linewidth = 1, color = '
plt.xlabel('lag difference')
plt.ylabel('lag-1 differenced solar output (kWh)')
plt.show()
```



The data now appears stationary with a constant mean and variance over time.

ACF and PACF plots

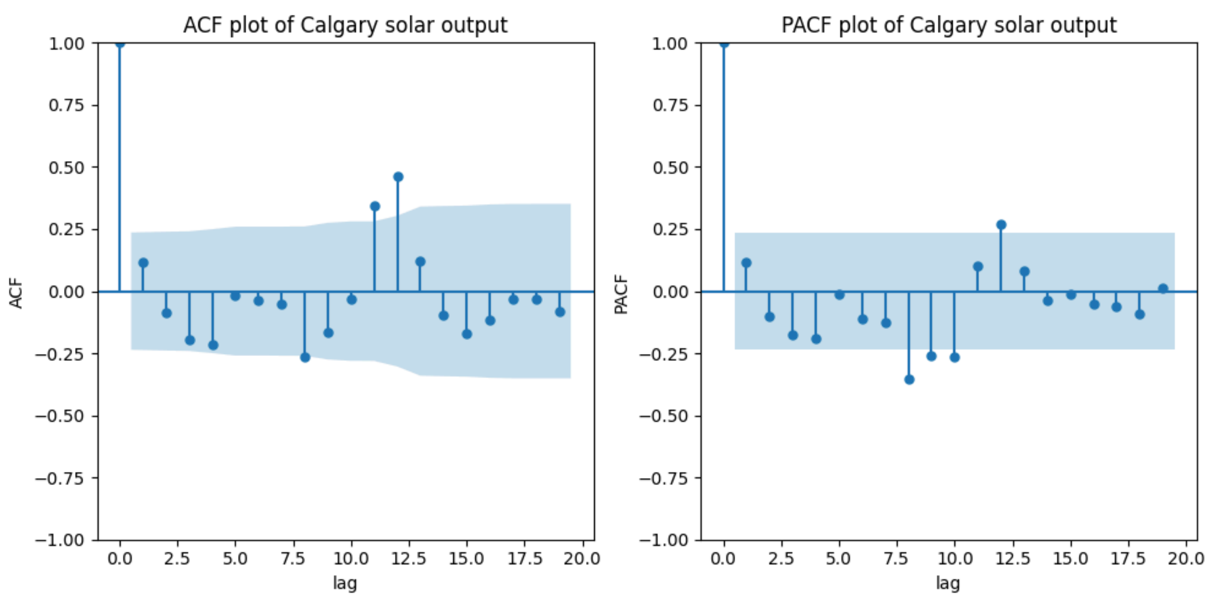
In [8]:
```
fig, axes = plt.subplots(nrows = 1, ncols = 2, figsize = (10, 5))

#create the ACF plot
plot_acf(df_diff['kWh'], title = 'ACF plot of Calgary solar output', ax = axes[0])

#create the PACF plot
plot_pacf(df_diff['kWh'], title = 'PACF plot of Calgary solar output', ax = axes[1]

#set the axes titles
axes[0].set_xlabel('lag')
axes[0].set_ylabel('ACF')
axes[1].set_xlabel('lag')
axes[1].set_ylabel('PACF')

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



ACF and PACF plots now show a lack of dependence structure in the data suggesting the data is now stationary.

Dickey-Fuller test for stationarity

In [9]:
```
#complete the Dickey-Fuller test
adfuller_test = adfuller(df_diff['kWh'])
print(f"ADF Statistic: {adfuller_test[0]:.4f}")
print(f"p-value: {adfuller_test[1]:.4f}")
```

```
ADF Statistic: -2.9672
p-value: 0.0381
```

The p-value is now <0.05 so the null hypothesis is rejected in favour of the alternative that the data is stationary. Therefore, 1st-order differencing has made the data stationary.

SARIMA model fitting

In a nutshell, an ARIMA(p, q, d) model is a linear regression model on previous p-lag values and previous q-lag errors post differencing d times.

Seasonal ARIMA (SARIMA) on the other hand are the additional set of parameters that specifically describe the seasonal components of the model. P, D, and Q represent the seasonal regression, differencing, and moving average coefficients, and m represents the number of data points in each seasonal cycle.

SARIMA(p,d,q)(P,D,Q)m where:

- p = order of autoregressive component
- d = degree of differencing
- q = order of moving average component
- P = seasonal order of autoregressive component
- D = seasonal degree of differencing
- Q = seaonal degree of moving average component
- m = number of observations per season

Train and test data split

In [10]:
```
#split the data into train and test sets
df_train = df[df['date'] < '2022-01-01']['kWh']
df_test = df[df['date'] >= '2022-01-01']['kWh']
```

Stepwise model optimization

The *pmdarima* library contains a function *auto_arima*, which helps identify the most optimal p, d, q, P, D, Q parameters and return a fitted ARIMA model.

In [11]:
```
#use stepwise algorithm to optimize the SARIMA model
sarima_model = auto_arima(df_train,
                          seasonal = True,
                          m = 12,
                          suppress_warnings = True,
                          stepwise = True,
                          trace = True,
                          njobs = -1)
sarima_model.fit(df_train)

#summarize the most optimal SARIMA model
sarima_model.summary()
```

```
Performing stepwise search to minimize aic
 ARIMA(2,0,2)(1,0,1)[12] intercept   : AIC=inf, Time=0.41 sec
 ARIMA(0,0,0)(0,0,0)[12] intercept   : AIC=553.123, Time=0.01 sec
 ARIMA(1,0,0)(1,0,0)[12] intercept   : AIC=512.084, Time=0.14 sec
 ARIMA(0,0,1)(0,0,1)[12] intercept   : AIC=522.880, Time=0.22 sec
 ARIMA(0,0,0)(0,0,0)[12]             : AIC=632.741, Time=0.01 sec
 ARIMA(1,0,0)(0,0,0)[12] intercept   : AIC=524.936, Time=0.05 sec
 ARIMA(1,0,0)(2,0,0)[12] intercept   : AIC=510.979, Time=0.28 sec
 ARIMA(1,0,0)(2,0,1)[12] intercept   : AIC=inf, Time=0.43 sec
 ARIMA(1,0,0)(1,0,1)[12] intercept   : AIC=510.892, Time=0.23 sec
 ARIMA(1,0,0)(0,0,1)[12] intercept   : AIC=517.502, Time=0.12 sec
 ARIMA(1,0,0)(1,0,2)[12] intercept   : AIC=513.908, Time=0.77 sec
 ARIMA(1,0,0)(0,0,2)[12] intercept   : AIC=inf, Time=0.26 sec
 ARIMA(1,0,0)(2,0,2)[12] intercept   : AIC=inf, Time=1.17 sec
 ARIMA(0,0,0)(1,0,1)[12] intercept   : AIC=527.414, Time=0.25 sec
 ARIMA(2,0,0)(1,0,1)[12] intercept   : AIC=512.893, Time=0.28 sec
 ARIMA(1,0,1)(1,0,1)[12] intercept   : AIC=512.885, Time=0.29 sec
 ARIMA(0,0,1)(1,0,1)[12] intercept   : AIC=516.331, Time=0.23 sec
 ARIMA(2,0,1)(1,0,1)[12] intercept   : AIC=inf, Time=0.32 sec
 ARIMA(1,0,0)(1,0,1)[12]             : AIC=517.184, Time=0.11 sec

Best model:  ARIMA(1,0,0)(1,0,1)[12] intercept
Total fit time: 5.589 seconds
```

Out[11]:

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	50
Model:	SARIMAX(1, 0, 0)x(1, 0, [1], 12)	**Log Likelihood**	-250.446
Date:	Thu, 18 Apr 2024	**AIC**	510.892
Time:	01:26:53	**BIC**	520.452
Sample:	0	**HQIC**	514.532
	- 50		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	3.6283	8.713	0.416	0.677	-13.448	20.705
ar.L1	0.5798	0.139	4.176	0.000	0.308	0.852
ar.S.L12	0.9272	0.172	5.379	0.000	0.589	1.265
ma.S.L12	-0.6115	0.473	-1.294	0.196	-1.538	0.315
sigma2	1087.9550	248.419	4.380	0.000	601.063	1574.847

Ljung-Box (L1) (Q):	0.01	**Jarque-Bera (JB):**	7.94
Prob(Q):	0.91	**Prob(JB):**	0.02
Heteroskedasticity (H):	0.47	**Skew:**	0.84
Prob(H) (two-sided):	0.13	**Kurtosis:**	4.00

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

Optimal SARIMA model:

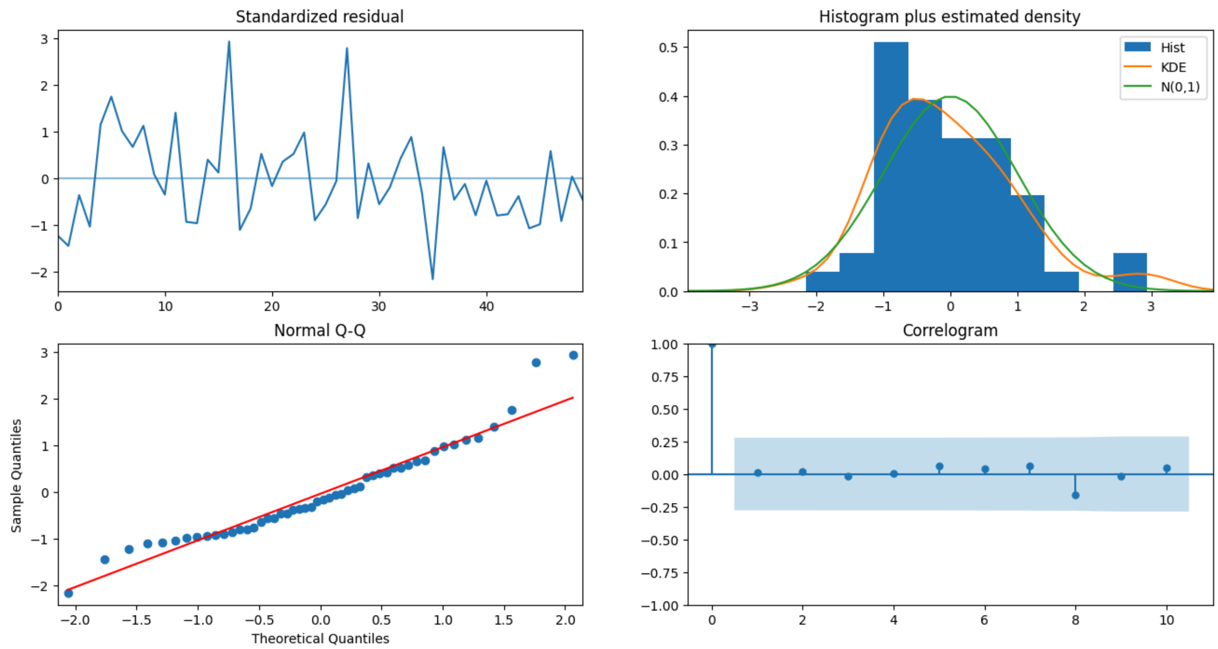
(p=1, d=0, q=0) (P=1, D=0, Q=1) m=12

Model validation

Model assumptions

In [12]:

```
#plot the model diagnostics
sarima_model.plot_diagnostics(figsize=(16, 8))
plt.show()
```



1. Stationarity: The top left plot shows the residuals over time, which appear to show a constant mean and variance. Additionally, the correlogram on the bottom right suggests that there is no autocorrelation in the residuals, and so they are effectively white noise. Therefore, the assumption of stationarity is met.

2. Homoscedasticity: Stationary data means constant variance across time, which supports the assumption of homoscedasticity.

3. Independence: Stationary data also means no temporal autocorrelation or dependence structure in the data, which supports the assumption of independence.

4. Normality: From the normal Q-Q plot, we can see that we almost have a straight line, which suggest no systematic departure from normality. The top-right plot shows that kde line (in orange) closely follows the N(0,1) standard normal distribution line (normal distribution with zero mean and standard deviation of 1), which confirms that the residuals are normally distributed.

Model forecasting

In [13]:
```
#make model predictions using test data
predictions, conf_int = sarima_model.predict(n_periods = len(df_test), return_conf_
```

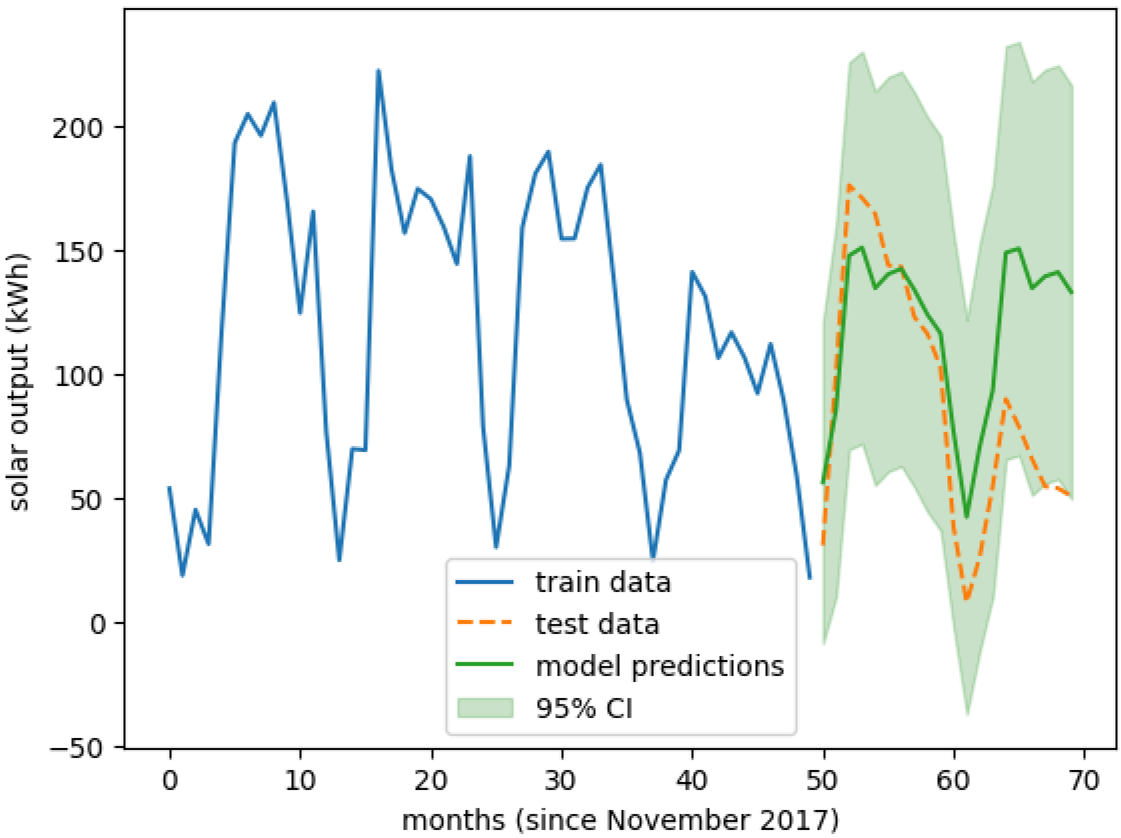
In [14]:
```
#plot the training data
plt.plot(df_train, label = 'train data')

#plot the testing data
plt.plot(df_test, label = 'test data', linestyle = '--')

#plot the model predictions
plt.plot(predictions, label = 'model predictions')
```

```
#plot the model prediction CI's
plt.fill_between(df_test.index,
                 conf_int[:, 0],
                 conf_int[:, 1],
                 color = 'green',
                 alpha = 0.2,
                 label = '95% CI')

#set the plot axis labels
plt.ylabel('solar output (kWh)')
plt.xlabel('months (since November 2017)')
plt.legend()
plt.show()
```



Essentially all of the test data points are contained within the predicted 95% confidence interval of forecasts. The model forecast mirros the test data well for the year 2022 but seems to deviate slightly for the year 2023. This may be due to an unexplained anomoly in 2023 resulting in relatively low levels of solar output compared to previous years.

Conclusion

The SARIMA model accurately models the average monthly solar outputs (in kWh) and seasonal patterns from the *Bearspaw Water Treatment Plant* photovoltaic site in the City of Calgary.