Problem Statement: -

Solar power consumption has been recorded by city councils at regular intervals. The reason behind doing so is to understand how businesses are using solar power so that they can cut down on nonrenewable sources of energy and shift towards renewable energy. Based on the data, build a forecasting model and provide insights on it.

In [1]:

```
#importing required libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:

```
solar = pd.read_csv("D:/DataScience/Class/assignment working/Forcasting/solarpower_cumuldaybyday2.csv")
```

In [3]:

```
solar["date"] = pd.to_datetime(solar["date"],infer_datetime_format = True)
indexed_data = solar.set_index(["date"])
```

In [4]:

```
from datetime import datetime
indexed_data.head(10)
```

Out[4]:

date	cum_power
2011-10-26	0.1
2011-10-27	10.2
2011-10-28	20.2
2011-10-29	29.6
2011-10-30	34.2
2011-10-31	38.0
2011-11-01	46.6
2011-11-02	51.6
2011-11-03	58.6
2011-11-04	60.5

In [5]:

```
indexed_data.tail()
```

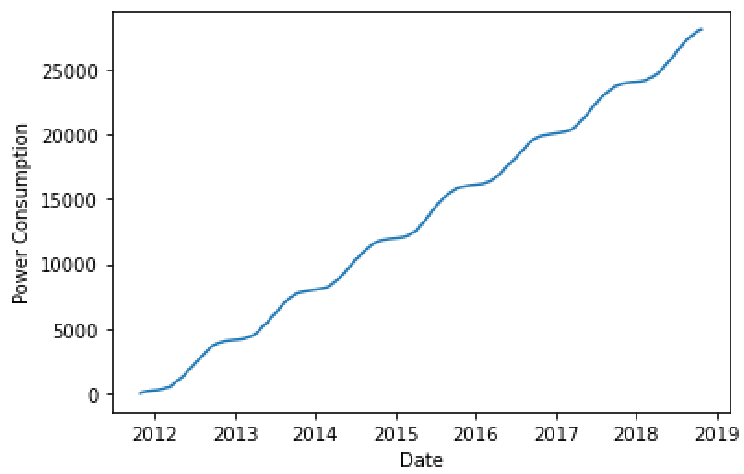
Out[5]:

date	cum_power
2018-10-22	28101.0
2018-10-23	28109.0
2018-10-24	28115.0
2018-10-25	28117.0
2018-10-26	28120.0

as the data is from 2011-10-26 to 2018 -10 -26 the cycle of 360 days startrts at 26th and ends at next 36th so no need to drop any rows

In [6]:

```
#plotting Graph
plt.xlabel("Date")
plt.ylabel("Power Consumption")
plt.plot(indexed_data)
plt.show()
```



Cheking Stationarity of data

Cheking Stationarity of data

ths is the default requirenment for any forcasting algorithm ,the data should be stationary

In [7]:
```
pd.options.display.max_rows = 999
```

In [8]:
```
pd.options.display.max_columns = 100
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [10]:
```
#checking stationarity with dicky-fuller test
from statsmodels.tsa.stattools import adfuller

dftest = adfuller(indexed_data["cum_power"], autolag = "AIC")
adfoutput = pd.Series(dftest[0:4],index = ["test statastic", "p_value","lag_used","number of observations used"])
for key,value in dftest[4].items():
    adfoutput["Critical value (%s)"%key]=value

print(adfoutput)
```

```
test statastic                  -0.421484
p_value                          0.906449
lag_used                        20.000000
number of observations used    2537.000000
Critical value (1%)             -3.432930
Critical value (5%)             -2.862680
Critical value (10%)            -2.567377
dtype: float64
```

we can not reject null hypothesis so data is not stationary

null hypothesis = data is not stationary

In [ ]:

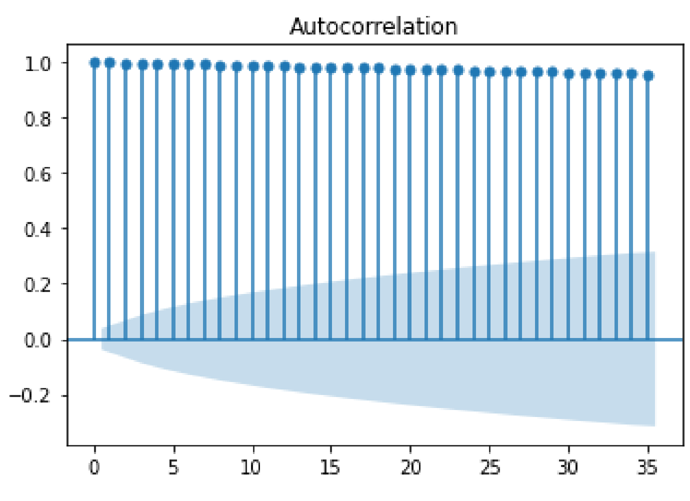
In [12]:
```
solar.isna().sum()
```

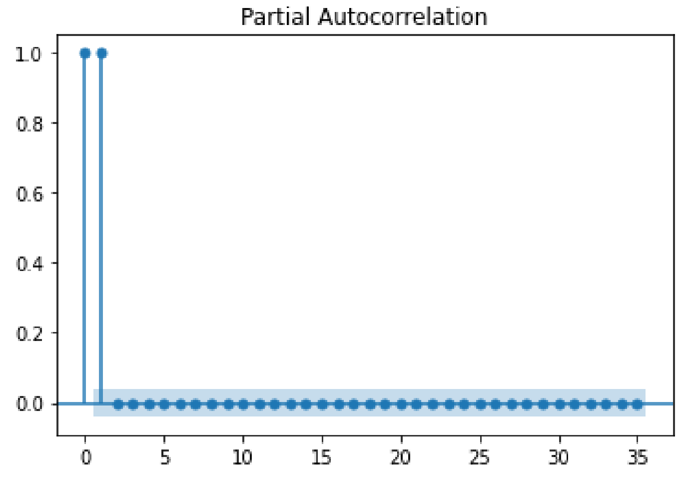
Out[12]:
```
date         0
cum_power    0
dtype: int64
```

In [14]:
```
from statsmodels.graphics.tsaplots import plot_acf ,plot_pacf
```

In [15]:
```
#checking accuracy with different lags and differentiation and moving average
```

In [18]:
```
#normal acf and pacf plot
plot_acf(indexed_data)
plot_pacf(indexed_data)
plt.show()
```





In [20]:

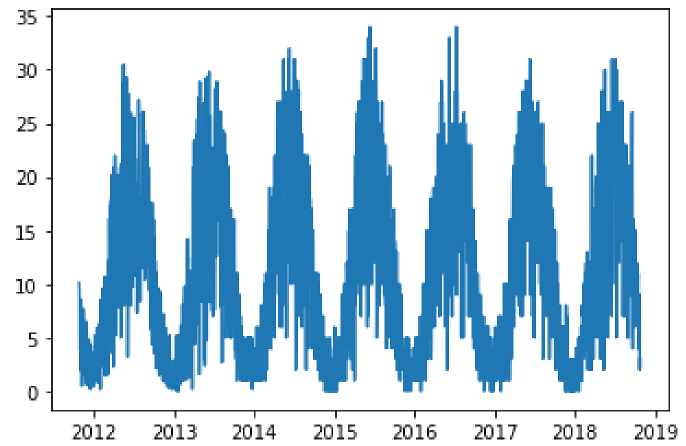
```
# first differencing
indexed_data["first_diff"]= indexed_data["cum_power"]-indexed_data["cum_power"].shift(1)
```

In [33]:

```
plt.plot(indexed_data.first_diff)
```

Out[33]:

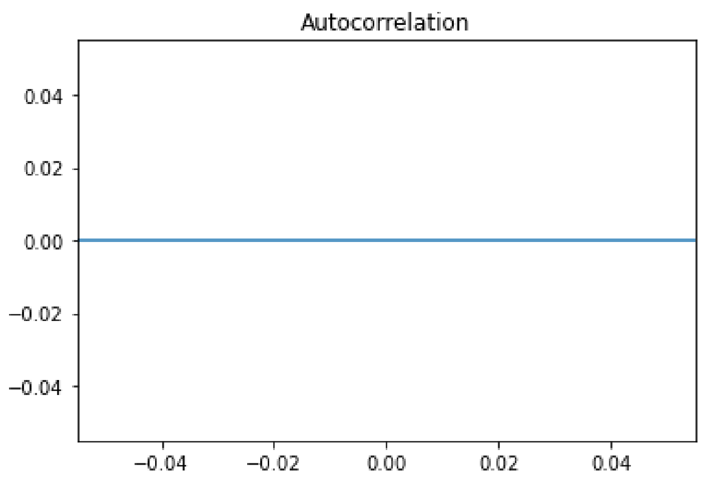
```
[<matplotlib.lines.Line2D at 0x208dcd0d730>]
```

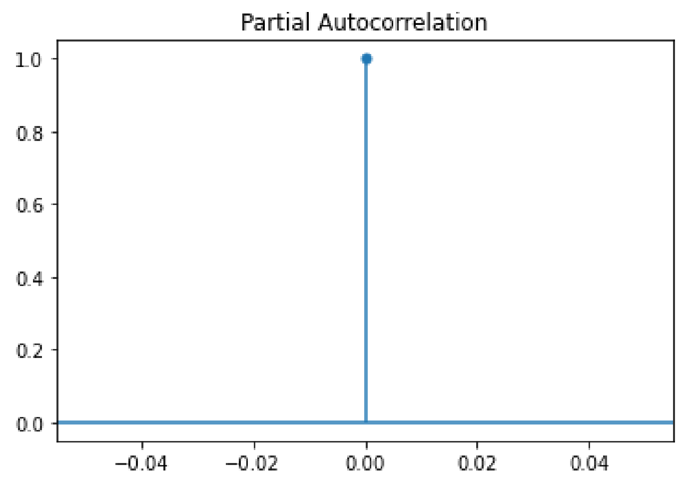


In [34]:

```
plot_acf(indexed_data.first_diff)
plot_pacf(indexed_data.first_diff)
plt.show()
```

```
C:\Users\theas\anaconda3\lib\site-packages\numpy\core\_asarray.py:83: UserWarning: Warning: converting a masked element to nan.
  return array(a, dtype, copy=False, order=order)
```





In [36]:

```
adfuller(indexed_data.first_diff.dropna())
```

Out[36]:

```
(-3.0856724129568414,
 0.02763182430737851,
 19,
 2537,
 {'1%': -3.4329301847920486,
  '5%': -2.862679919243664,
  '10%': -2.5673768219208686},
 15175.367039787136)
```

In [41]:

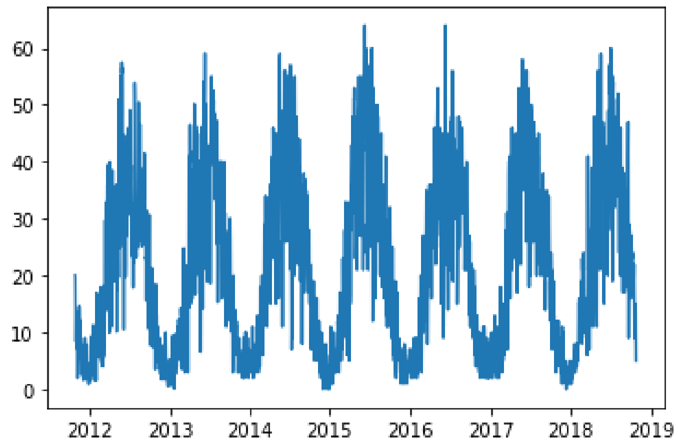
```
# second differencing
indexed_data["second_diff"]= indexed_data["cum_power"]-indexed_data["cum_power"].shift(1).shift(1)
```

In [42]:

```
plt.plot(indexed_data["second_diff"])
```

Out[42]:

```
[<matplotlib.lines.Line2D at 0x208dc8b00d0>]
```



In [43]:

```
indexed_data["second_diff"].head()
```

Out[43]:

```
date
2011-10-26     NaN
2011-10-27     NaN
2011-10-28    20.1
2011-10-29    19.4
2011-10-30    14.0
Name: second_diff, dtype: float64
```

In [44]:

```
adfuller(indexed_data.second_diff.dropna())
```

Out[44]:

```
(-2.965202630211606,
 0.0382743102354102,
 26,
 2529,
 {'1%': -3.432938355012086,
  '5%': -2.8626835272597217,
  '10%': -2.567378742868999},
 15251.943584943776)
```

In [45]:

```
indexed_data["second_diff"]
```

Out[45]:

```
date
2011-10-26     NaN
2011-10-27     NaN
2011-10-28    20.1
2011-10-29    19.4
2011-10-30    14.0
              ...
2018-10-22    15.0
2018-10-23    14.0
2018-10-24    14.0
2018-10-25     8.0
2018-10-26     5.0
Name: second_diff, Length: 2558, dtype: float64
```

In [ ]:

In [53]:

```
#pip install pmdarima
```

In [55]:

```
from pmdarima import auto_arima
```

In [56]:

```
auto_model=auto_arima(indexed_data.cum_power,trace=True) ### returns best p,d,q values based on AIC score
```

```
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=15320.006, Time=2.34 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=18025.637, Time=0.07 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=15947.568, Time=0.30 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=16895.797, Time=0.43 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=20652.126, Time=0.05 sec
 ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=15319.616, Time=1.15 sec
 ARIMA(0,1,2)(0,0,0)[0] intercept   : AIC=16517.081, Time=0.69 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=15379.388, Time=0.88 sec
```

```
 ARIMA(1,1,3)(0,0,0)[0] intercept   : AIC=15320.898, Time=1.48 sec
 ARIMA(0,1,3)(0,0,0)[0] intercept   : AIC=16248.109, Time=1.18 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=15318.019, Time=1.27 sec
 ARIMA(2,1,0)(0,0,0)[0] intercept   : AIC=15701.892, Time=0.43 sec
 ARIMA(3,1,1)(0,0,0)[0] intercept   : AIC=15320.013, Time=1.87 sec
 ARIMA(3,1,0)(0,0,0)[0] intercept   : AIC=15535.761, Time=0.67 sec
 ARIMA(3,1,2)(0,0,0)[0] intercept   : AIC=15320.532, Time=3.22 sec
 ARIMA(2,1,1)(0,0,0)[0]             : AIC=15323.620, Time=0.64 sec

Best model:  ARIMA(2,1,1)(0,0,0)[0] intercept
Total fit time: 16.754 seconds
```

In [57]:

```
auto_model.summary()
```

Out[57]:

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	2558
Model:	SARIMAX(2, 1, 1)	**Log Likelihood**	-7654.009
Date:	Sun, 04 Jul 2021	**AIC**	15318.019
Time:	14:48:18	**BIC**	15347.252
Sample:	0	**HQIC**	15328.620
	- 2558		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	0.0635	0.044	1.447	0.148	-0.023	0.149
ar.L1	1.1941	0.020	59.228	0.000	1.155	1.234
ar.L2	-0.2001	0.019	-10.373	0.000	-0.238	-0.162
ma.L1	-0.8640	0.012	-73.828	0.000	-0.887	-0.841
sigma2	23.2928	0.582	40.034	0.000	22.152	24.433

Ljung-Box (L1) (Q):	0.00	**Jarque-Bera (JB):**	60.60
Prob(Q):	1.00	**Prob(JB):**	0.00
Heteroskedasticity (H):	1.10	**Skew:**	0.01
Prob(H) (two-sided):	0.17	**Kurtosis:**	3.75

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

In [171]:

```
df=pd.read_csv("D:/DataScience/Class/assignment working/Forcasting/solarpower_cumuldaybyday2.csv")
```

Building ARIMA model

In [174]:

```
Train = df.head(2192)
Test=df.tail(366)
```

In [175]:

```
# Creating a function to calculate the MAPE value for test data
def MAPE(pred,org):
    temp = np.abs((pred-org)/org)*100
    return np.mean(temp)
```

In [184]:

```
# Simple Exponential Method
from statsmodels.tsa.holtwinters import SimpleExpSmoothing
ses_model = SimpleExpSmoothing(Train["cum_power"]).fit(smoothing_level=1)
pred_ses = ses_model.predict(start = Test.index[0], end = Test.index[-1])
MAPE(pred_ses, Test.cum_power)
```

Out[184]:

```
5.922982997009587
```

In [189]:

```
np.sqrt(np.mean(pred_ses-Test.cum_power)**2)
```

Out[189]:

```
1584.4754098360656
```

In [186]:

```
# Holt method
from statsmodels.tsa.holtwinters import Holt
hw_model = Holt(Train["cum_power"]).fit(smoothing_level=0.01)
pred_hw = hw_model.predict(start = Test.index[0], end = Test.index[-1])
MAPE(pred_hw, Test.cum_power)
```

```
C:\Users\theas\anaconda3\lib\site-packages\statsmodels\tsa\holtwinters\model.py:427: FutureWarning: After 0.13 initialization must
```

```
be handled at model creation
  warnings.warn(
C:\Users\theas\anaconda3\lib\site-packages\statsmodels\tsa\holtwinters\model.py:920: ConvergenceWarning: Optimization failed to co
nverge. Check mle_retvals.
  warnings.warn(
```

Out[186]:

```
2.327441434347866
```

In [188]:

```python
np.sqrt(np.mean(pred_hw-Test.cum_power)**2)
```

Out[188]:

```
576.2850881654938
```

In [193]:

```python
# Holts winter exponential smoothing with additive seasonality and additive trend
from statsmodels.tsa.holtwinters import ExponentialSmoothing
hwe_model_add_add = ExponentialSmoothing(Train["cum_power"], seasonal = "add", trend = "add", seasonal_periods = 2).fit(smoothing
_level=0.01)
pred_hwe_add_add = hwe_model_add_add.predict(start = Test.index[0], end = Test.index[-1])
MAPE(pred_hwe_add_add, Test.cum_power)
```

Out[193]:

```
2.0681153138377666
```

In [195]:

```python
np.sqrt(np.mean(pred_hwe_add_add-Test.cum_power)**2)
```

Out[195]:

```
427.2307160122235
```

In [198]:

```python
# Holts winter exponential smoothing with multiplicative seasonality and additive trend
hwe_model_mul_add = ExponentialSmoothing(Train["cum_power"], seasonal = "mul", trend = "add", seasonal_periods = 2).fit(smoothing
_level=0.01)
pred_hwe_mul_add = hwe_model_mul_add.predict(start = Test.index[0], end = Test.index[-1])
MAPE(pred_hwe_mul_add, Test.cum_power)
```

Out[198]:

```
1.9882712951755088
```

In [199]:

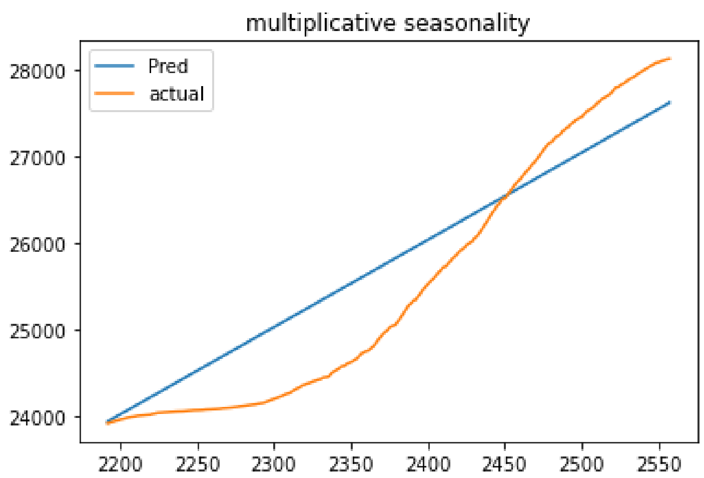
```python
np.sqrt(np.mean(pred_hwe_mul_add-Test.cum_power)**2)
```

Out[199]:

```
275.19585296625985
```

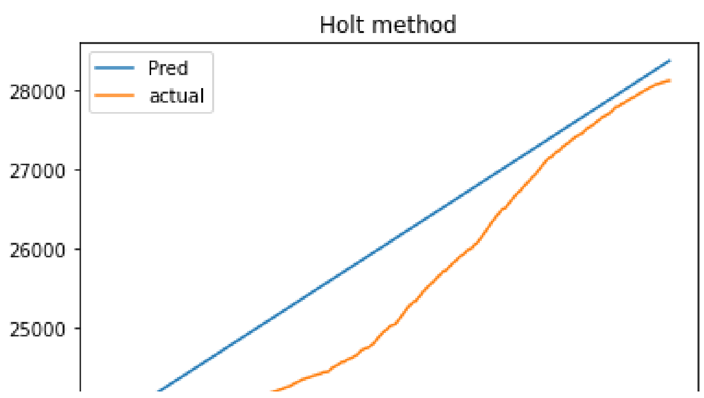
In [203]:

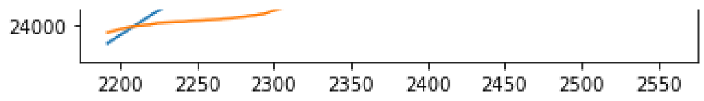
```python
plt.plot(pred_hwe_mul_add,label="Pred")
plt.plot(Test.cum_power,label="actual")
plt.title("multiplicative seasonality")
plt.legend()
plt.show()
```



In [204]:

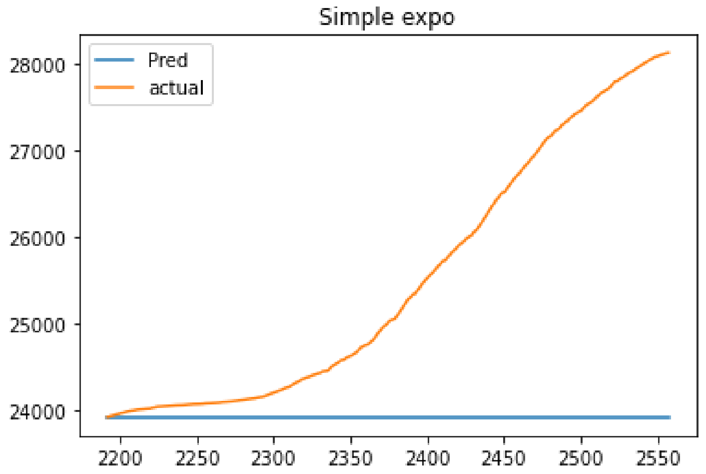
```python
plt.plot(pred_hw,label="Pred")
plt.plot(Test.cum_power,label="actual")
plt.title("Holt method")
plt.legend()
plt.show()
```





In [205]:

```
plt.plot(pred_ses,label="Pred")
plt.plot(Test.cum_power,label="actual")
plt.title("Simple expo")
plt.legend()
plt.show()
```



Final Model

predicting for next one year

In [206]:

```
pred = pd.read_csv("D:/DataScience/Class/assignment working/Forcasting/pred_solarpower_cumuldaybyday2.csv")
```

In [221]:

```
final_mod = ExponentialSmoothing(solar["cum_power"], seasonal = "mul", trend = "add", seasonal_periods = 2).fit()
```

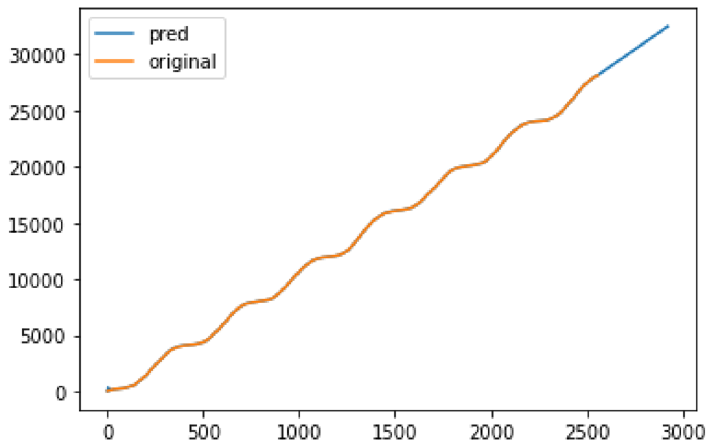
```
C:\Users\theas\anaconda3\lib\site-packages\statsmodels\tsa\holtwinters\model.py:920: ConvergenceWarning: Optimization failed to co
nverge. Check mle_retvals.
  warnings.warn(
```

In [222]:

```
final_pred =final_mod.predict(start = pred.index[0], end = pred.index[-1])
```

In [223]:

```
plt.plot(final_pred,label="pred")
plt.plot(solar.cum_power,label="original")
plt.legend()
plt.show()
```



In [224]:

```
import statsmodels.api as sm
```

In [225]:

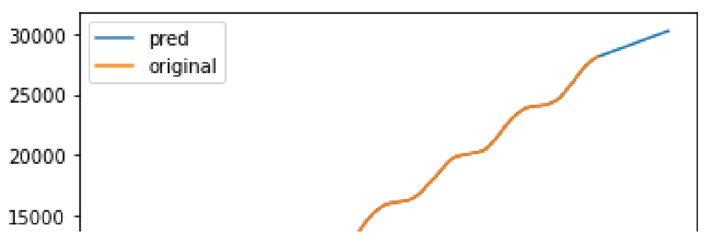
```
model = sm.tsa.statespace.SARIMAX(solar.cum_power,order=(1,1,1),seasonal_order=(1,1,1,2)) #seasonal_order=(p,d,q,m) #m=lag
results=model.fit()
```

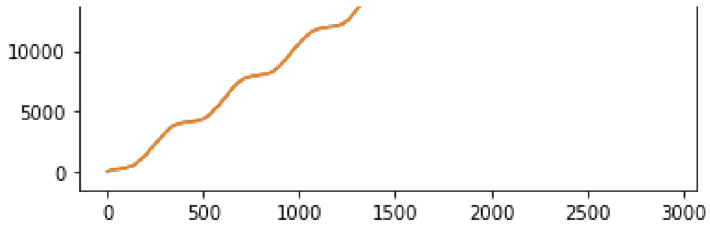
In [226]:

```
predic=results.predict(start = pred.index[0], end = pred.index[-1])
```

In [227]:

```
plt.plot(predic,label="pred")
plt.plot(solar.cum_power,label="original")
plt.legend()
plt.show()
```





Summary and inference

Im anablw to solve this problem ,as you can see the results are pathetic, i need model solution for this problem from your side,thank you

```
In [ ]:
```