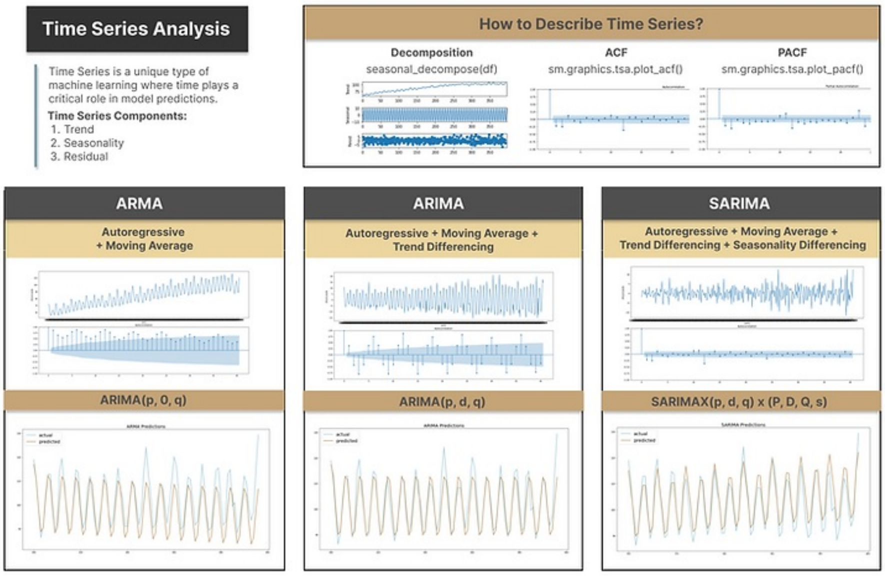
Energy Price Prediction with ARIMA & SARIMA



In [1]:
```
!pip install ta
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: ta in c:\users\pytho\appdata\roaming\python\python311\site-packages (0.11.0)
Requirement already satisfied: numpy in c:\users\pytho\appdata\roaming\python\python311\site-packages (from ta)
(1.26.4)
Requirement already satisfied: pandas in c:\users\pytho\appdata\roaming\python\python311\site-packages (from ta
) (2.1.4)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\programdata\anaconda3\lib\site-packages (from panda
s->ta) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\programdata\anaconda3\lib\site-packages (from pandas->ta) (20
23.3.post1)
Requirement already satisfied: tzdata>=2022.1 in c:\programdata\anaconda3\lib\site-packages (from pandas->ta) (
2023.3)
Requirement already satisfied: six>=1.5 in c:\programdata\anaconda3\lib\site-packages (from python-dateutil>=2.
8.2->pandas->ta) (1.16.0)
```

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime

from scipy import stats
import statsmodels.api as sm
from itertools import product

import plotly.graph_objects as go
from ta.trend import MACD
from ta.momentum import RSIIndicator
from ta.volatility import BollingerBands
from statsmodels.tsa.arima.model import ARIMA
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error

import warnings
warnings.filterwarnings("ignore")

sns.set(rc={"axes.facecolor":"Beige" , "axes.grid" : False})
```

In [3]:
```
df = pd.read_csv("/kaggle/input/european-union-energy-market-data/EU_energy_data.csv")
df.head()
```

Out[3]:

	Unnamed: 0	fecha	hora	sistema	bandera	precio	tipo_moneda	origen_dato	fecha_actualizacion
0	0	2010-07-21	1	HU	1	39.287	1	6	2021-10-01 12:39:53
1	1	2010-07-21	2	HU	1	35.925	1	6	2021-10-01 12:39:53
2	2	2010-07-21	3	HU	1	33.223	1	6	2021-10-01 12:39:53
3	3	2010-07-21	4	HU	1	30.842	1	6	2021-10-01 12:39:53
4	4	2010-07-21	5	HU	1	33.395	1	6	2021-10-01 12:39:53

In [4]:
```
df = df.rename(columns = {'fecha' : 'Date','hora' : 'Hour' , 'sistema' : 'EU_countries','bandera' : 'Renewable/
```

```
                          'precio' : 'Cost(€/MWh)','tipo_moneda' : 'CurrencyType','origen_dato' : 'DataSource',
df = df.drop('Unnamed: 0',axis=1)

#df['Date'] = pd.to_datetime(df['Date'], format='%Y-%m-%d')
df['Hour'] = df['Hour'].astype(str).str.zfill(2)

try:
  df['Hour'] = pd.to_numeric(df['Hour'])
except:
  # Handle conversion errors (e.g., non-numeric characters)
  print("Error converting 'Hour' column to numeric")

# Function to convert the range
def convert_range(value):
  # Handle edge cases (leading zero and exceeding 24)
  if value == '01':
    return 0
  elif value > 24:
    raise ValueError("Value exceeds 24")
  else:
    # Remove leading zero (assuming strings) or subtract 1 (assuming integers)
    return int(value) - 1 if isinstance(value, int) else int(value[1:])

# Apply the conversion function
df['Hour'] = df['Hour'].apply(convert_range)

# Function to replace values with leading zeros (handles all cases)
def replace_with_leading_zero(value):
  if 0 <= value <= 23:
    return f"{value:02d}"  # Use f-string for consistent formatting
  else:
    raise ValueError(f"Value {value} is outside the range 0-12")

# Apply the function
df['Hour'] = df['Hour'].apply(replace_with_leading_zero)
df['Hour'] = df['Hour'].astype(str)  # Ensure Hour is string type
df['Hour'] = df['Hour'] + ':00:00'

df["Period"] = df[["Date","Hour"]].apply(" ".join, axis=1)

df = df [['Period','EU_countries', 'Renewable/Non_Renewable',
       'Cost(€/MWh)', 'CurrencyType', 'DataSource', 'Updated_Date']]

df['Period'] = pd.to_datetime(df['Period'],format ="%Y-%m-%d %H:%M:%S" )
df.head()
```

Out[4]:

	Period	EU_countries	Renewable/Non_Renewable	Cost(€/MWh)	CurrencyType	DataSource	Updated_Date
0	2010-07-21 00:00:00	HU	1	39.287	1	6	2021-10-01 12:39:53
1	2010-07-21 01:00:00	HU	1	35.925	1	6	2021-10-01 12:39:53
2	2010-07-21 02:00:00	HU	1	33.223	1	6	2021-10-01 12:39:53
3	2010-07-21 03:00:00	HU	1	30.842	1	6	2021-10-01 12:39:53
4	2010-07-21 04:00:00	HU	1	33.395	1	6	2021-10-01 12:39:53

In [5]:

```
df['EU_countries'] = df['EU_countries'].replace('FR', 'France')
df.head()
```

Out[5]:

	Period	EU_countries	Renewable/Non_Renewable	Cost(€/MWh)	CurrencyType	DataSource	Updated_Date
0	2010-07-21 00:00:00	HU	1	39.287	1	6	2021-10-01 12:39:53
1	2010-07-21 01:00:00	HU	1	35.925	1	6	2021-10-01 12:39:53
2	2010-07-21 02:00:00	HU	1	33.223	1	6	2021-10-01 12:39:53
3	2010-07-21 03:00:00	HU	1	30.842	1	6	2021-10-01 12:39:53
4	2010-07-21 04:00:00	HU	1	33.395	1	6	2021-10-01 12:39:53

In [6]:

```
df_France = df[df['EU_countries']== "France"]
df_France.head()
```

Out[6]:

	Period	EU_countries	Renewable/Non_Renewable	Cost(€/MWh)	CurrencyType	DataSource	Updated_Date
179166	2014-01-01 00:00:00	France	0	15.15	1	1	2021-10-01 12:39:53
179187	2014-01-01 01:00:00	France	0	12.96	1	1	2021-10-01 12:39:53
179208	2014-01-01 02:00:00	France	0	12.09	1	1	2021-10-01 12:39:53
179229	2014-01-01 03:00:00	France	0	11.70	1	1	2021-10-01 12:39:53
179250	2014-01-01 04:00:00	France	0	11.66	1	1	2021-10-01 12:39:53

In [7]:

```
# Remove the unnecessary feature
df_France = df_France.drop(['EU_countries','CurrencyType','DataSource','Updated_Date','Renewable/Non_Renewable'
df_France.head().style.set_properties(subset=['Period'], **{'background-color': 'yellow'})
```

Out[7]:

	Period	Cost(€/MWh)
179166	2014-01-01 00:00:00	15.150000
179187	2014-01-01 01:00:00	12.960000
179208	2014-01-01 02:00:00	12.090000
179229	2014-01-01 03:00:00	11.700000
179250	2014-01-01 04:00:00	11.660000

In [8]:

```
France_df = df_France.copy('Deep')
```

In [9]:

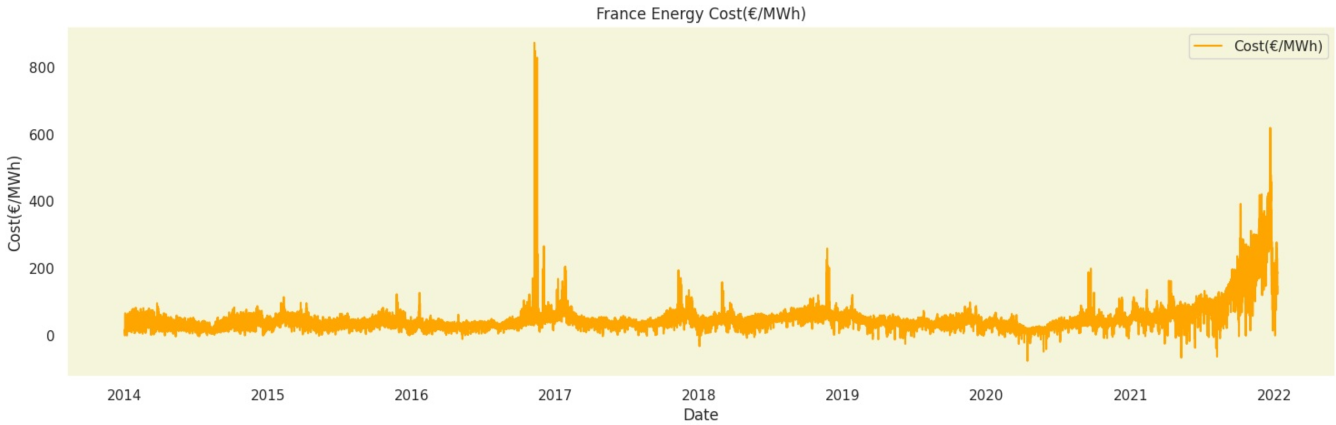
```
df_France = df_France.set_index('Period')
df_France.sort_index(inplace=True)
df_France.head()
```

Out[9]:

Period	Cost(€/MWh)
2014-01-01 00:00:00	15.15
2014-01-01 01:00:00	12.96
2014-01-01 02:00:00	12.09
2014-01-01 03:00:00	11.70
2014-01-01 04:00:00	11.66

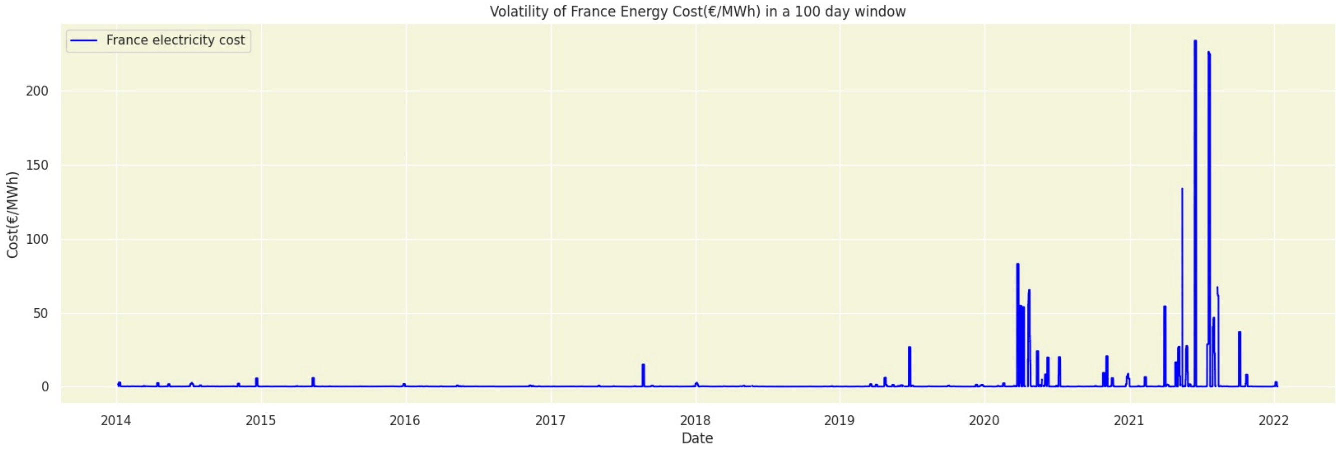
In [10]:

```
plt.figure(figsize = (18,5))
plt.plot(df_France.index,df_France['Cost(€/MWh)'],label= 'Cost(€/MWh)',color = 'orange')
plt.title('France Energy Cost(€/MWh)')
plt.xlabel('Date')
plt.ylabel('Cost(€/MWh)')
plt.grid(False)
plt.legend()
plt.show()
```

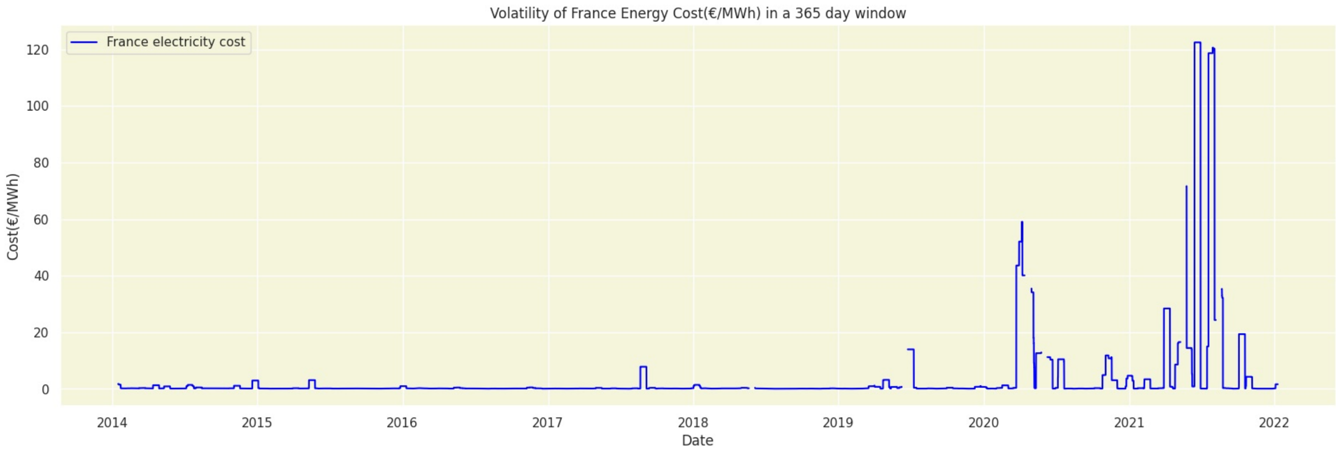


In [11]:

```
df_France['Change'] = df_France['Cost(€/MWh)'].pct_change()
window_size = 100
df_100 = df_France['Change'].rolling(window = window_size).std()


#plotting
plt.figure(figsize = (20,6))
plt.plot(df_France.index , df_100 , label = 'France electricity cost' , color = 'blue')
plt.title('Volatility of France Energy Cost(€/MWh) in a 100 day window')
plt.xlabel('Date')
plt.ylabel('Cost(€/MWh)')
plt.legend()
plt.grid(True)
plt.show()
```

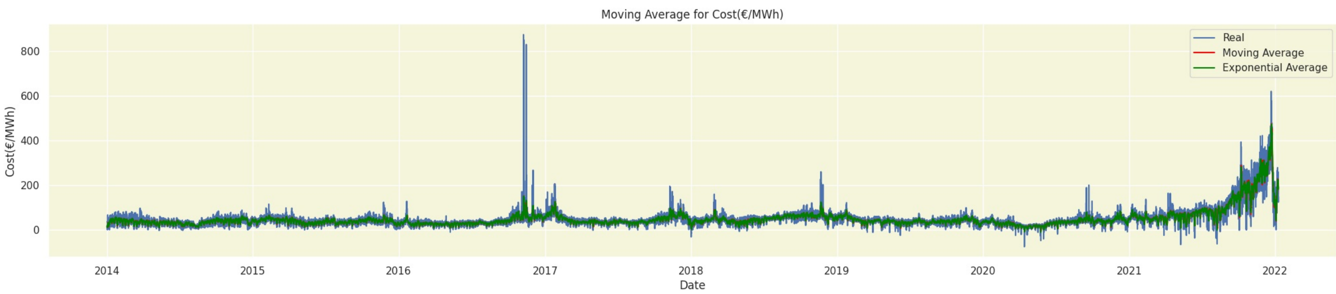


In [12]:
```
df_France['Change'] = df_France['Cost(€/MWh)'].pct_change()
window_size = 365
df_365 = df_France['Change'].rolling(window = window_size).std()


#plotting
plt.figure(figsize = (20,6))
plt.plot(df_France.index , df_365 , label = 'France electricity cost' , color = 'blue')
plt.title('Volatility of France Energy Cost(€/MWh) in a 365 day window')
plt.xlabel('Date')
plt.ylabel('Cost(€/MWh)')
plt.legend()
plt.grid(True)
plt.show()
```



In [13]:
```
df_France['MA'] = df_France['Cost(€/MWh)'].rolling(window = 30).mean()
df_France['EMA'] = df_France['Cost(€/MWh)'].ewm(span = 30 , adjust = False).mean()
```

In [14]:
```
plt.figure(figsize = (25,10))

plt.subplot(2,1,1)
plt.plot(df_France.index,df_France['Cost(€/MWh)'], label = 'Real' )
plt.plot(df_France.index,df_France['MA'] , label = 'Moving Average' , color = 'red')
plt.plot(df_France.index,df_France['EMA'] , label = 'Exponential Average' , color = 'green')
plt.title('Moving Average for Cost(€/MWh)')
plt.xlabel('Date')
plt.ylabel('Cost(€/MWh)')
plt.legend()
plt.grid()
```



In [15]:
```
France_df.head()
```

Out[15]:

	Period	Cost(€/MWh)
179166	2014-01-01 00:00:00	15.15
179187	2014-01-01 01:00:00	12.96
179208	2014-01-01 02:00:00	12.09
179229	2014-01-01 03:00:00	11.70
179250	2014-01-01 04:00:00	11.66

In [16]:

```
France_df = France_df.set_index('Period')
France_df.head()
```

Out[16]:

	Cost(€/MWh)
Period	
2014-01-01 00:00:00	15.15
2014-01-01 01:00:00	12.96
2014-01-01 02:00:00	12.09
2014-01-01 03:00:00	11.70
2014-01-01 04:00:00	11.66

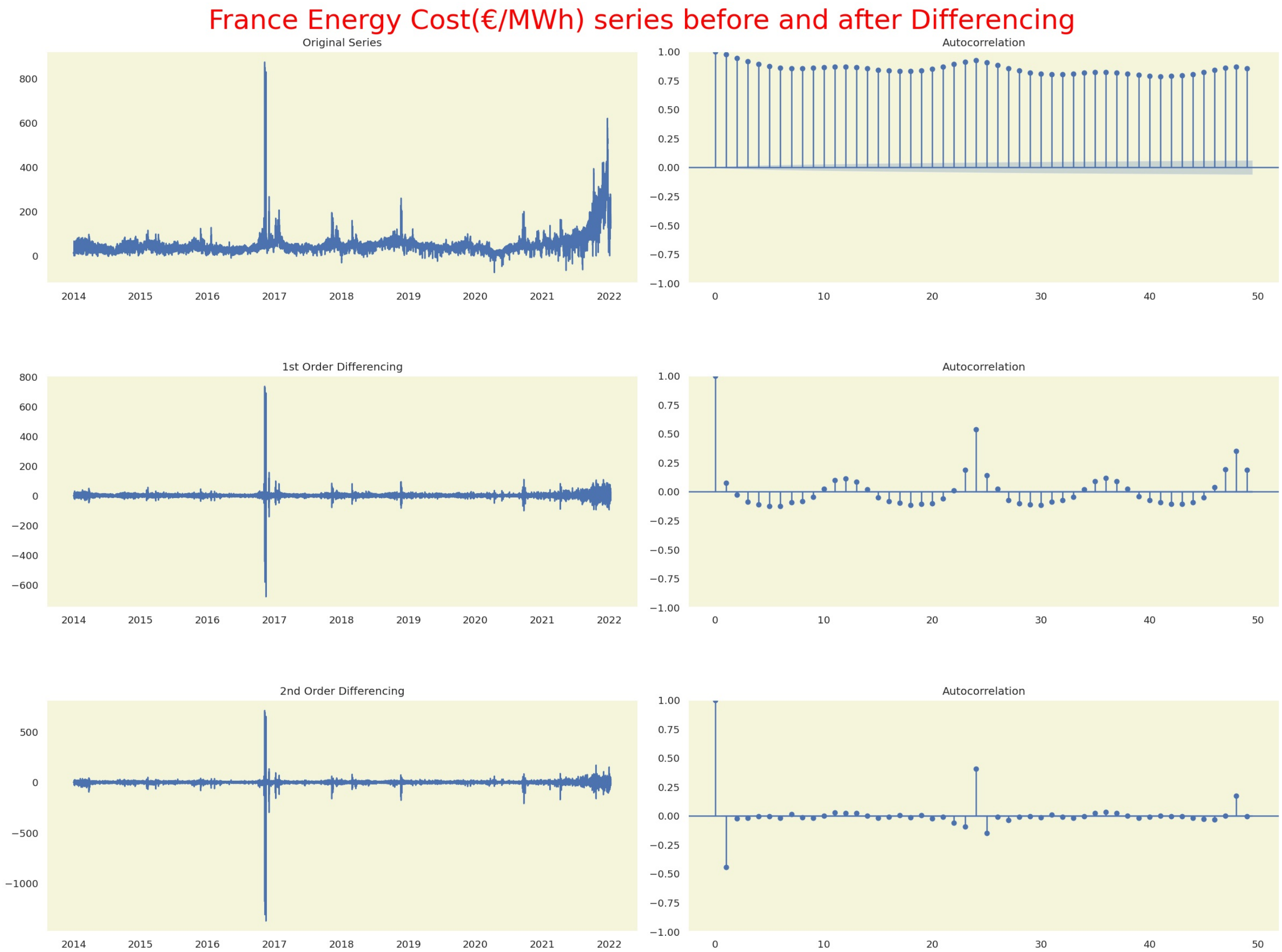
In [17]:

```
from statsmodels.tsa.stattools import adfuller
from numpy import log
result = adfuller(France_df['Cost(€/MWh)'])
print('ADF Statistic: %f' % result[0])
print('p-value: %f' % result[1])
print("\n")
print(result)
```

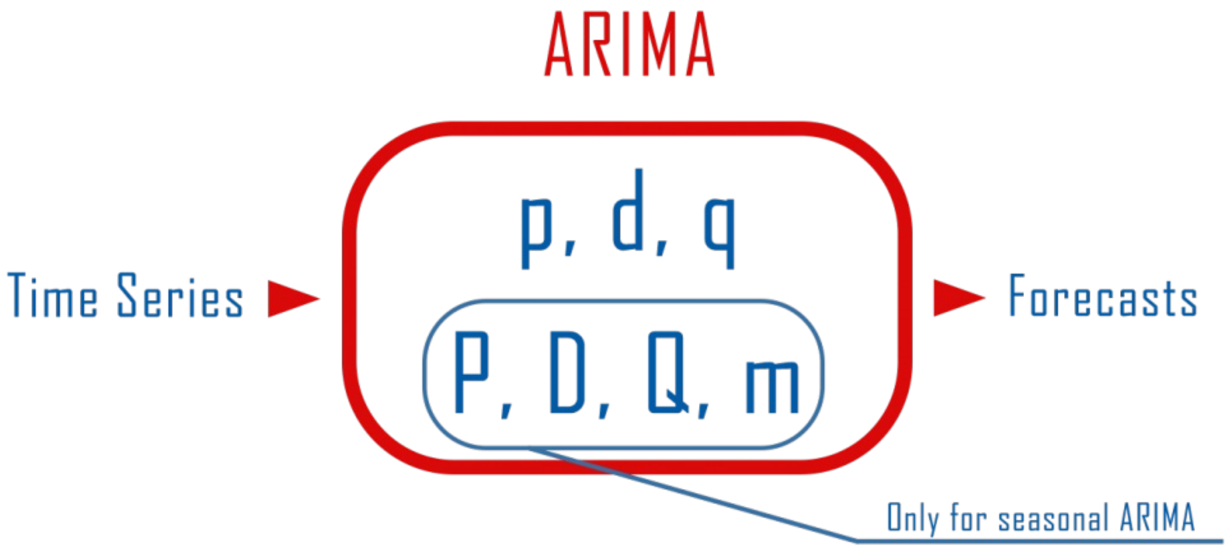
```
ADF Statistic: -7.237777
p-value: 0.000000


(-7.237777344012639, 1.9193771760542327e-10, 59, 70260, {'1%': -3.430443076272825, '5%': -2.8615811380624825, '
10%': -2.5667918963845726}, 482550.7669923681)
```

In [18]:

```
from statsmodels.graphics.tsaplots import plot_acf

plt.rcParams.update({'figure.figsize':(20,15), 'figure.dpi':120})  # Adjust the dimensions as needed.

# Original Series
fig, axes = plt.subplots(3, 2)
axes[0, 0].plot(France_df['Cost(€/MWh)'])
axes[0, 0].set_title('Original Series')

# Plotting the ACF
plot_acf(France_df['Cost(€/MWh)'], ax=axes[0, 1])

# 1st Differencing
axes[1, 0].plot(France_df['Cost(€/MWh)'].diff())
axes[1, 0].set_title('1st Order Differencing')
plot_acf(France_df['Cost(€/MWh)'].diff().dropna(), ax=axes[1, 1])

# 2nd Differencing
axes[2, 0].plot(France_df['Cost(€/MWh)'].diff().diff())
axes[2, 0].set_title('2nd Order Differencing')
plot_acf(France_df['Cost(€/MWh)'].diff().diff().dropna(), ax=axes[2, 1])

plt.suptitle('France Energy Cost(€/MWh) series before and after Differencing', size = 30,color= 'Red')
plt.tight_layout()

fig.subplots_adjust(hspace=0.4)

plt.show()
```



ARIMA Model



ARIMA: A Time Machine for Predictions

ARIMA, standing for Autoregressive Integrated Moving Average, is a statistical model that empowers you to forecast future values based on past observations in a time series dataset. It's a popular choice for analyzing and predicting trends in various fields, from finance (stock prices) to weather forecasting (temperature variations).

The Building Blocks of ARIMA:

ARIMA breaks down the time series into three key components:

1. Autoregressive (AR): This captures the influence of past values on the current value. Imagine predicting tomorrow's stock price; ARIMA considers the closing prices from the past few days (up to a specified order 'p') to make a prediction.

2. Integrated (I): Sometimes, data exhibits non-stationarity, meaning its characteristics (mean, variance) change over time. Differencing, a technique that removes the trend by subtracting the previous value from the current value, is applied (up to order 'd') to achieve stationarity.

3. Moving Average (MA): This component considers the impact of past forecast errors (up to order 'q') on the current prediction. Essentially, it incorporates the idea that errors from past predictions might influence future errors, helping to refine the forecast.

Understanding the Notation:

ARIMA models are represented using the notation ARIMA(p, d, q), where:

- p signifies the number of autoregressive terms (past values considered).
- d indicates the degree of differencing needed to achieve stationarity.
- q represents the number of moving average terms (past forecast errors considered).

Example: Predicting Sales Figures

Suppose you're a business analyst tasked with forecasting monthly sales. You might use an ARIMA model to analyze historical sales data. Here's a breakdown of a possible scenario:

- ARIMA(2, 1, 1): This model considers the influence of the past two months' sales figures (p=2) and incorporates differencing once (d=1) to remove any trends. Additionally, it takes into account the error from the previous month's forecast (q=1) to refine the current prediction.

Strengths of ARIMA:

1. Effective for Stationary Data: ARIMA excels at analyzing and forecasting time series data that exhibits stationarity.
2. Relatively Straightforward Implementation: Compared to more complex models, ARIMA offers a good balance between accuracy and interpretability.
3. Wide Range of Applications: Its flexibility makes it applicable in various domains like finance, economics, and environmental science.

Limitations to Consider:

1. Stationarity is a Must: The model's effectiveness relies heavily on the data being stationary. If not, transformations like differencing might be necessary.
2. Parameter Tuning is Crucial: Choosing the optimal values for p, d, and q can be an iterative process. Techniques like examining the autocorrelation function (ACF) and partial autocorrelation function (PACF) can assist in this selection.
3. Limited for Non-linear Relationships: ARIMA assumes a linear relationship between past observations and future values. If the relationship is non-linear, other models might be more suitable.

In conclusion, ARIMA serves as a powerful tool for time series forecasting, particularly when dealing with stationary data. Its interpretability and diverse applications make it a cornerstone technique in the data scientist's toolkit.

In [19]:
```
France_df.head()
```

Out[19]:

	Cost(€/MWh)
Period	
2014-01-01 00:00:00	15.15
2014-01-01 01:00:00	12.96
2014-01-01 02:00:00	12.09
2014-01-01 03:00:00	11.70
2014-01-01 04:00:00	11.66

In [20]:
```
result = adfuller(France_df.diff().dropna())
```

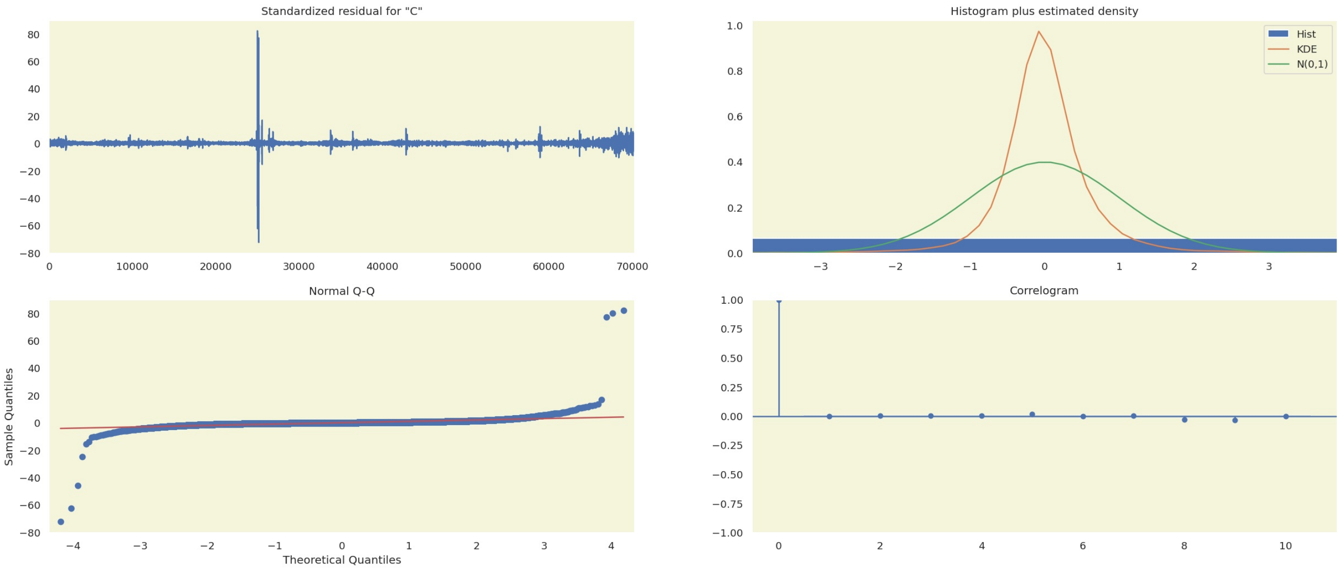
In [21]:
```
result = adfuller(France_df.diff()['Cost(€/MWh)'].dropna())
print('ADF Statistic: %f' % result[0])
print('p-value: %f' % result[1])
```

```
ADF Statistic: -43.063020
p-value: 0.000000
```

The value is lesser than the significance level of 0.05 and hence the series is stationary

In [22]:
```
from statsmodels.tsa.arima.model import ARIMA

train_size = int(len(France_df)*0.8)
train , test = France_df.iloc[:train_size] , France_df.iloc[train_size:]

p = 5
d = 1
q = 2

model = ARIMA(France_df, order=(p, d, q))
model_fit = model.fit()

# Summary of the model
print(model_fit.summary())

# Plot diagnostic plots
model_fit.plot_diagnostics(figsize=(25,10))
plt.show()
```

```
                               SARIMAX Results                                
==============================================================================
Dep. Variable:             Cost(€/MWh)   No. Observations:                70320
Model:                  ARIMA(5, 1, 2)   Log Likelihood             -254800.742
Date:                 Fri, 17 May 2024   AIC                         509617.484
Time:                         14:22:17   BIC                         509690.770
Sample:                              0   HQIC                        509640.082
                               - 70320                                         
Covariance Type:                   opg                                         
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          1.1478      0.043     26.799      0.000       1.064       1.232
ar.L2         -0.3220      0.039     -8.288      0.000      -0.398      -0.246
ar.L3         -0.0324      0.004     -7.569      0.000      -0.041      -0.024
ar.L4         -0.0138      0.003     -4.883      0.000      -0.019      -0.008
ar.L5         -0.0344      0.004     -8.951      0.000      -0.042      -0.027
ma.L1         -1.1693      0.043    -27.304      0.000      -1.253      -1.085
ma.L2          0.2318      0.040      5.831      0.000       0.154       0.310
sigma2        82.1950      0.015   5527.591      0.000      82.166      82.224
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):       17148130175.63
Prob(Q):                              0.96   Prob(JB):                         0.00
Heteroskedasticity (H):               3.91   Skew:                            11.68
Prob(H) (two-sided):                  0.00   Kurtosis:                      2422.12
===================================================================================

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

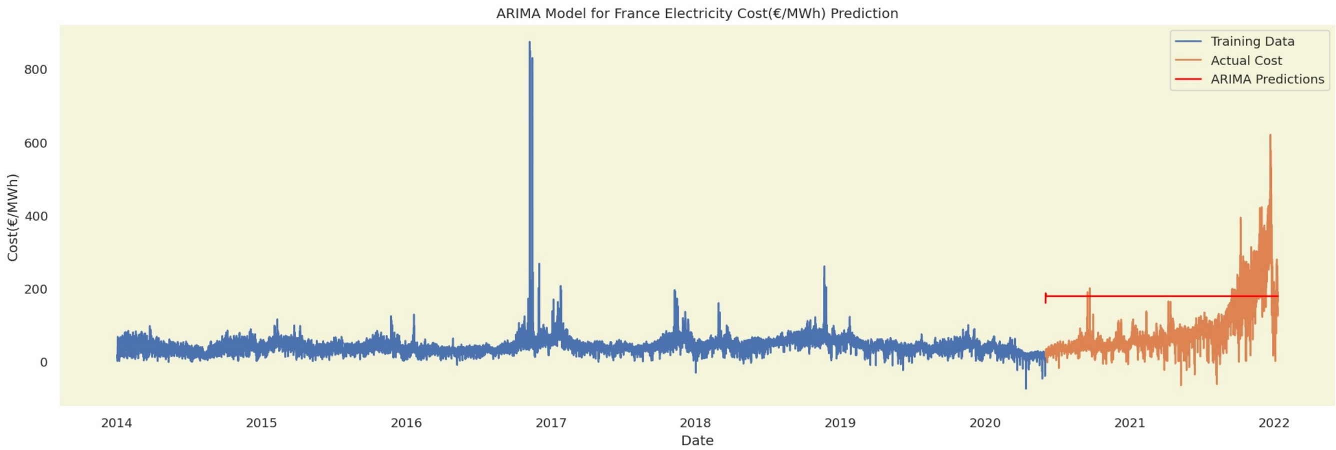


In [23]:
```
# Make predictions on the test set
predictions = model_fit.forecast(steps=len(test))

# Evaluate the model
mse = mean_squared_error(test, predictions)
print(f'Mean Squared Error: {mse}')
```

```
Mean Squared Error: 14578.36492162369
```

In [24]:
```
# Plot the results
plt.figure(figsize=(20, 6))
plt.plot(train, label='Training Data')
plt.plot(test, label='Actual Cost')
plt.plot(test.index, predictions, label='ARIMA Predictions', color='red')
```

```python
plt.title('ARIMA Model for France Electricity Cost(€/MWh) Prediction')
plt.xlabel('Date')
plt.ylabel('Cost(€/MWh)')
plt.legend()
plt.show()
```



SARIMA Model

SARIMA for Time Series Forecasting

In the realm of time series analysis, predicting future values from past observations is a constant pursuit. SARIMA (Seasonal Autoregressive Integrated Moving Average) emerges as a powerful tool for this task, particularly for data exhibiting seasonality.

What is SARIMA?

SARIMA is a statistical model that builds upon the ARIMA (Autoregressive Integrated Moving Average) model by incorporating a seasonal component. It leverages past observations (AR), potential differencing to achieve stationarity (I), and past errors (MA) to make predictions, all while accounting for seasonal patterns (seasonal AR, seasonal differencing, and seasonal MA).

Key Components of SARIMA:

- Non-seasonal AR (p): This captures the influence of past values (up to p lags) on the current value.
- Differencing (d): Differencing is applied if the data exhibits non-stationarity (trend or increasing/decreasing variance). It removes the trend by subtracting the previous value from the current value.
- Non-seasonal MA (q): This considers the impact of past forecast errors (up to q lags) on the current value.
- Seasonal AR (P): Similar to non-seasonal AR, this captures the influence of past seasonal values (e.g., past year's values for monthly data).
- Seasonal Differencing (D): Similar to differencing, this removes seasonal trends if present.
- Seasonal MA (Q): This considers the impact of past seasonal forecast errors on the current value.
- Seasonality (s): This specifies the number of periods in a single season (e.g., s=12 for monthly data).

$$SARIMA\underbrace{(p,d,q)}_{non-seasonal}\underbrace{(P,D,Q)_m}_{seasonal}$$

Example: Predicting Electricity Prices

Imagine you're tasked with forecasting electricity prices. Electricity usage often exhibits seasonality, with higher demand during peak summer and winter months. A SARIMA model can be a good fit for this scenario.

Here's a simplified example:

You might choose a model with p=2 (considering the influence of the past two days' prices), d=1 (differencing to remove trends), q=1 (accounting for the previous day's forecast error). Additionally, you might include seasonal components like P=1 (considering the influence of the same day last month's price) and s=12 (accounting for monthly seasonality). By analyzing past electricity prices and fitting the SARIMA model with these parameters, you can generate forecasts for future electricity prices.

Benefits of SARIMA:

- Effective for Seasonal Data: It excels at capturing and predicting trends in data with seasonal patterns.
- Relatively Easy to Implement: Compared to more complex models, SARIMA offers a good balance of accuracy and interpretability.
- Provides Model Diagnostics: The model can be assessed to identify potential shortcomings and refine its parameters for better results.

Limitations of SARIMA:

- Requires Stationary Data: The model assumes stationarity in the data. If trends or non-constant variance exist, differencing might be needed.
- Parameter Tuning Can Be Challenging: Choosing the optimal hyperparameters (p, d, q, P, D, Q, s) can be an iterative process. Techniques like ACF and PACF analysis can help guide this process.

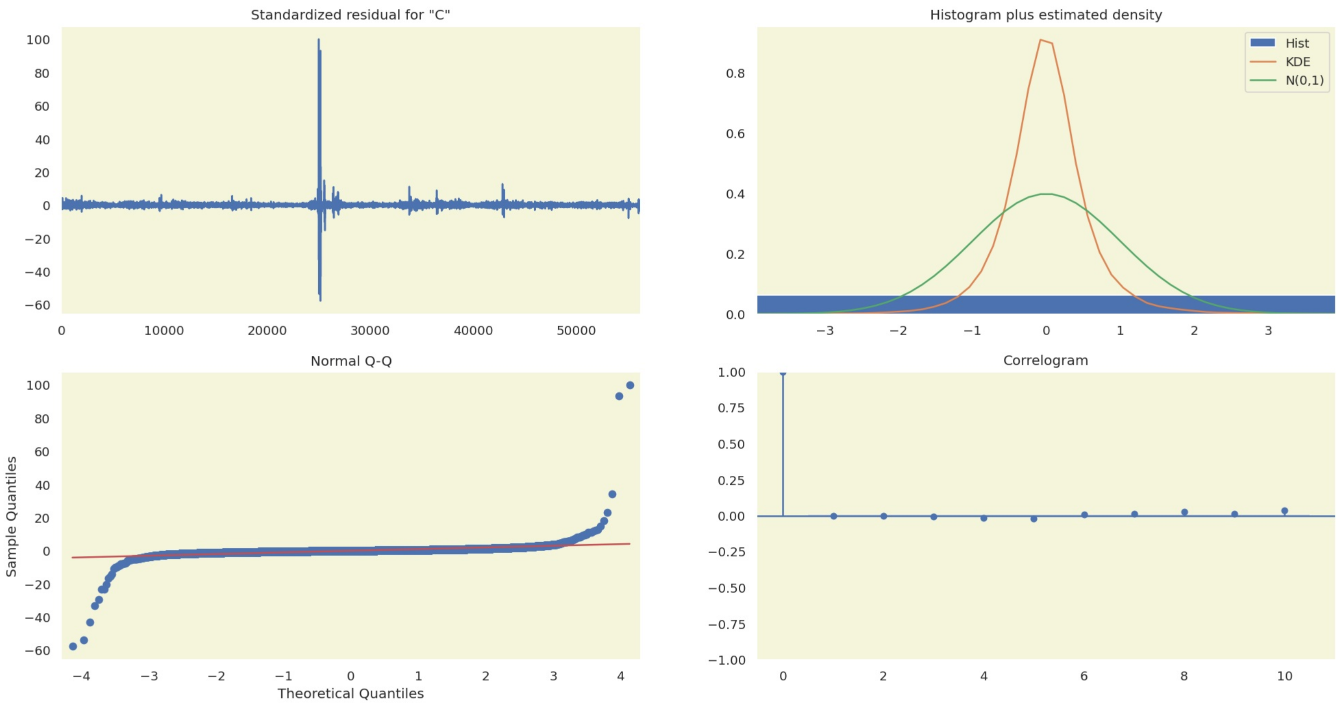
Overall, SARIMA is a versatile tool for time series forecasting, particularly when dealing with seasonal data. Its interpretable nature and effectiveness make it a valuable weapon in the arsenal of data scientists and analysts.

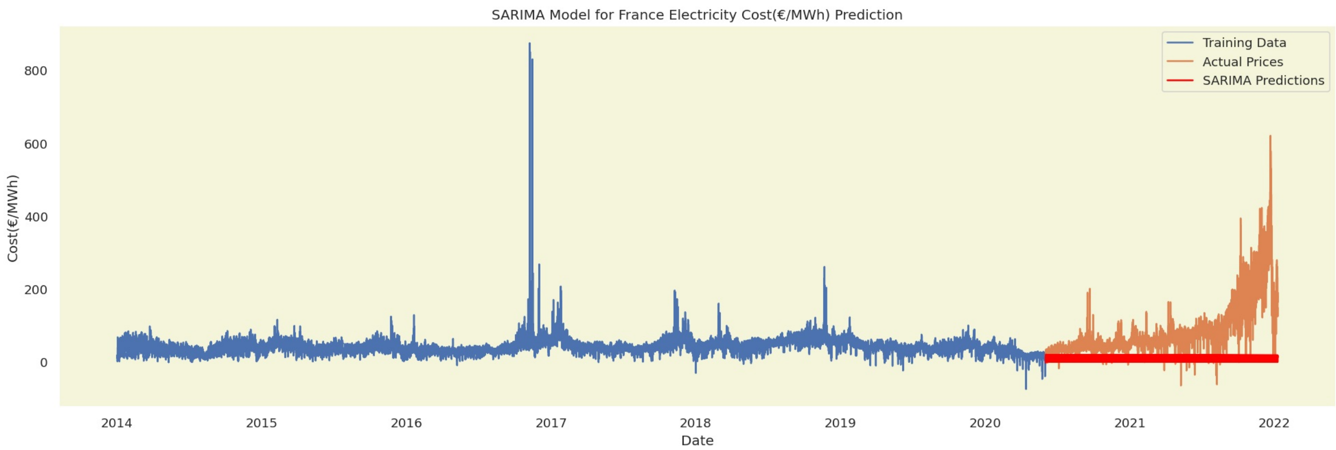
In [25]:

```python
from statsmodels.tsa.statespace.sarimax import SARIMAX
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

# Splitting data (assuming France_df is your data)
train_size = int(len(France_df) * 0.8)
train, test = France_df.iloc[:train_size], France_df.iloc[train_size:]

# Define hyperparameters in a dictionary (easier to modify)
model_params = {
    'order': (5, 1, 2),  # Non-seasonal parameters (p, d, q)
    'seasonal_order': (1, 1, 1, 12),  # Seasonal parameters (P, D, Q, s)
}

# Build and fit the model in one step
model = SARIMAX(train, **model_params).fit()

# Make predictions and calculate MSE
predictions = model.forecast(steps=len(test))
mse = mean_squared_error(test, predictions)

print(model.summary())

model.plot_diagnostics(figsize=(20, 10))
plt.show()

# Plot results (adjust figure size as needed)
plt.figure(figsize=(20, 6))
plt.plot(train, label='Training Data')
plt.plot(test, label='Actual Prices')
plt.plot(test.index, predictions, label='SARIMA Predictions', color='red')
plt.title('SARIMA Model for France Electricity Cost(€/MWh) Prediction')
plt.xlabel('Date')
plt.ylabel('Cost(€/MWh)')
plt.legend()
plt.show()
```

```
RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =           10     M =           10

At X0         0 variables are exactly at the bounds

At iterate    0    f=  3.60087D+00    |proj g|=  4.85907D-02
 This problem is unconstrained.
```

```
At iterate    5    f=  3.45920D+00    |proj g|=  4.35460D-02

At iterate   10    f=  3.44006D+00    |proj g|=  8.68927D-03

At iterate   15    f=  3.43883D+00    |proj g|=  4.24092D-03

At iterate   20    f=  3.43155D+00    |proj g|=  8.94043D-03

At iterate   25    f=  3.42888D+00    |proj g|=  4.30408D-03

At iterate   30    f=  3.42679D+00    |proj g|=  1.94234D-02

At iterate   35    f=  3.42472D+00    |proj g|=  2.36750D-03

At iterate   40    f=  3.42448D+00    |proj g|=  2.27587D-03

At iterate   45    f=  3.42443D+00    |proj g|=  7.90514D-04

At iterate   50    f=  3.42442D+00    |proj g|=  4.47846D-04

           * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value

           * * *

   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
   10     50     56      1     0     0   4.478D-04   3.424D+00
  F =   3.4244244303013112

STOP: TOTAL NO. of ITERATIONS REACHED LIMIT
                                      SARIMAX Results
============================================================================================
Dep. Variable:                          Cost(€/MWh)   No. Observations:                56256
Model:             SARIMAX(5, 1, 2)x(1, 1, [1], 12)   Log Likelihood             -192644.421
Date:                              Fri, 17 May 2024   AIC                         385308.842
Time:                                      14:27:39   BIC                         385398.216
Sample:                                           0   HQIC                        385336.685
                                            - 56256
Covariance Type:                                opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.3549      0.000  -1117.611      0.000      -0.356      -0.354
ar.L2          0.7073      0.000   1612.280      0.000       0.706       0.708
ar.L3          0.1321      0.001    178.201      0.000       0.131       0.134
ar.L4          0.1032      0.001    115.870      0.000       0.101       0.105
ar.L5          0.0704      0.001     77.746      0.000       0.069       0.072
ma.L1         -0.0003      0.450     -0.001      1.000      -0.883       0.882
ma.L2         -0.9997      0.450     -2.221      0.026      -1.882      -0.117
ar.S.L12      -0.6343      0.000  -2604.472      0.000      -0.635      -0.634
ma.S.L12      -0.2550      0.000   -539.411      0.000      -0.256      -0.254
sigma2        55.3010     24.892      2.222      0.026       6.513     104.089
===================================================================================
Ljung-Box (L1) (Q):                   0.02   Jarque-Bera (JB):       30770502346.89
Prob(Q):                              0.89   Prob(JB):                         0.00
Heteroskedasticity (H):               1.07   Skew:                            24.13
Prob(H) (two-sided):                  0.00   Kurtosis:                      3626.27
===================================================================================

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





Time Series Part : 1

1. Energy Price Prediction with LSTM-Time Series

2. Exploratory Data Analysis (EDA) for Time Series

3. Fb-Prophet with Multiple Regressors

4. PV-forecast with Neural-Prophet

5. Wind and Solar Power Generation Time Series

6. Europe Power Generation EDA & Solar TS-Analysis

7. Fb-Prophet Step by Step

8. Time Series Neuro-Prophet Step by Step

9. Time-series-analysis | fb-prophet-explained

10. Fb-Prophet High Accuracy with irregular data gaps

11. Day ahead electricity prices forecast with TS

12. Indian Power Consumption analysis with fb-prophet

Time Series Part : 2

13. Energy Price Prediction with ARIMA & SARIMA

14. Energy Price Prediction with RNN & LSTM

15. Energy Price Prediction with XGBoost-Time Series

16. Neural-Prophet Unlocking Tomorrow's energy Price

4. Energy Price Prediction with LSTM-Time Series

17. Energy Price Prediction with ARIMA

6. Anomaly Detection in energy cost with fb-Prophet

18. Energy Market's Day Ahead Prices prediction (EDA)

19. Day ahead electricity prices forecast with Lasso, LGBMRegressor & CatBoostRegressor