In [2]:
```python
# Manipulación y tratamiento de Datos
import numpy as np
import pandas as pd

# Visualización de datos
import plotly.express as px
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('ggplot')

# Modelación Arima
from statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.stattools import adfuller

# Métrica de Evaluación
from sklearn.metrics import mean_squared_error
from statsmodels.tools.eval_measures import rmse
from sklearn import metrics

# No presentar advertencias
import warnings
warnings.filterwarnings("ignore")
```

In [3]:
```python
import tensorflow as tf
```

In [4]:
```python
from keras.preprocessing.sequence import TimeseriesGenerator
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
```

In [5]:
```python
df = pd.read_csv('temperaturesbarcelonadesde1780.csv')
df.head()
```

Out[5]:

	Any	Temp_Mitjana_Gener	Temp_Mitjana_Febrer	Temp_Mitjana_Marc	Temp_Mitjana_Abril	Temp_Mitjana_Maig
0	1786	7.8	8.3	9.9	12.8	16.8
1	1787	5.4	7.8	11.3	12.1	14.7
2	1788	6.4	10.1	10.4	12.5	17.1
3	1789	6.9	9.3	8.7	13.3	17.7
4	1790	7.4	9.5	10.4	12.3	15.0

In [6]:
```python
df.describe()
```

Out[6]:

	Any	Temp_Mitjana_Gener	Temp_Mitjana_Febrer	Temp_Mitjana_Marc	Temp_Mitjana_Abril	Temp_Mi
count	237.000000	237.000000	237.000000	237.000000	237.000000	
mean	1904.000000	7.671308	8.616878	10.308861	12.487342	
std	68.560193	1.506237	1.640887	1.409609	1.243086	
min	1786.000000	3.400000	2.500000	6.100000	9.400000	
25%	1845.000000	6.700000	7.700000	9.500000	11.600000	
50%	1904.000000	7.700000	8.600000	10.300000	12.500000	
75%	1963.000000	8.900000	9.600000	11.100000	13.300000	

max	2022.000000	10.700000	12.800000	14.800000	15.900000

In [7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 237 entries, 0 to 236
Data columns (total 13 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   Any                     237 non-null    int64  
 1   Temp_Mitjana_Gener      237 non-null    float64
 2   Temp_Mitjana_Febrer     237 non-null    float64
 3   Temp_Mitjana_Marc       237 non-null    float64
 4   Temp_Mitjana_Abril      237 non-null    float64
 5   Temp_Mitjana_Maig       237 non-null    float64
 6   Temp_Mitjana_Juny       237 non-null    float64
 7   Temp_Mitjana_Juliol     237 non-null    float64
 8   Temp_Mitjana_Agost      237 non-null    float64
 9   Temp_Mitjana_Setembre   237 non-null    float64
 10  Temp_Mitjana_Octubre    237 non-null    float64
 11  Temp_Mitjana_Novembre   237 non-null    float64
 12  Temp_Mitjana_Desembre   237 non-null    float64
dtypes: float64(12), int64(1)
memory usage: 24.2 KB
```

In [8]:
```
#rename columns
df.rename(columns={'Temp_Mitjana_Gener': "1",
                   'Temp_Mitjana_Febrer': "2",
                   'Temp_Mitjana_Marc': "3",
                   'Temp_Mitjana_Abril': "4",
                   'Temp_Mitjana_Maig': "5",
                   'Temp_Mitjana_Juny': "6",
                   'Temp_Mitjana_Juliol': "7",
                   'Temp_Mitjana_Agost': "8",
                   'Temp_Mitjana_Setembre': "9",
                   'Temp_Mitjana_Octubre': "10",
                   'Temp_Mitjana_Novembre': "11",
                   'Temp_Mitjana_Desembre': "12"},inplace=True)
```

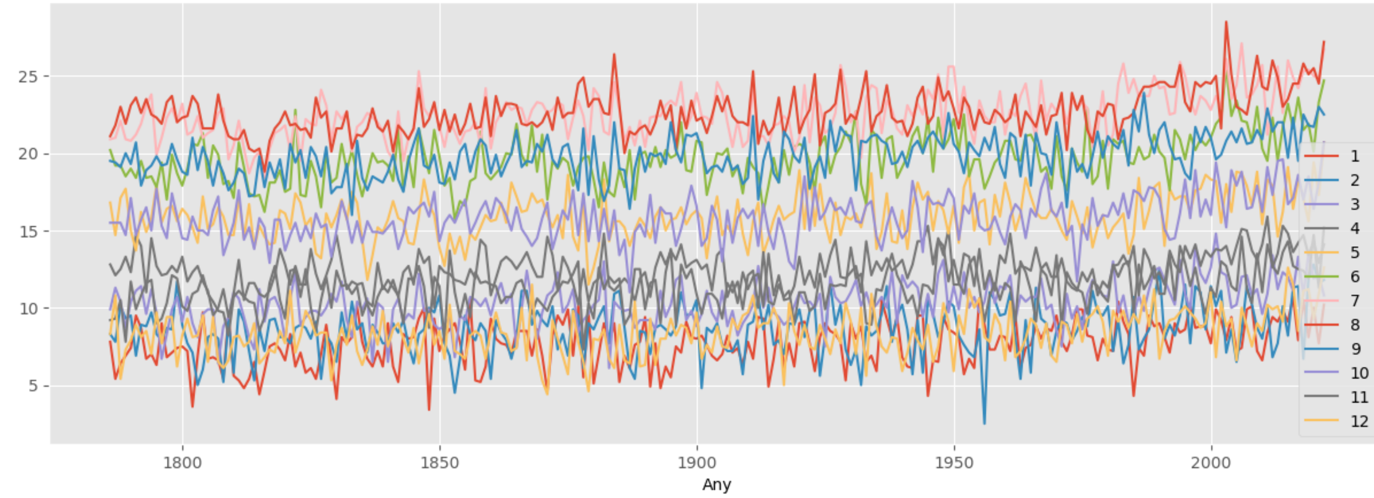
In [9]:
```
df.columns
```

Out[9]:
```
Index(['Any', '1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12'], dtype='obj
ect')
```

In [10]:
```
import seaborn as sns
```

In [11]:
```
df.plot(x= 'Any',figsize=(15, 5))
```

Out[11]:
```
<Axes: xlabel='Any'>
```



In [12]:
```
# Creates a pivot table dataframe
table = df.melt(id_vars=['Any'], value_vars=['1', '2', '3', '4', '5', '6', '7',
        '8', '9', '10', '11', '12'])
```

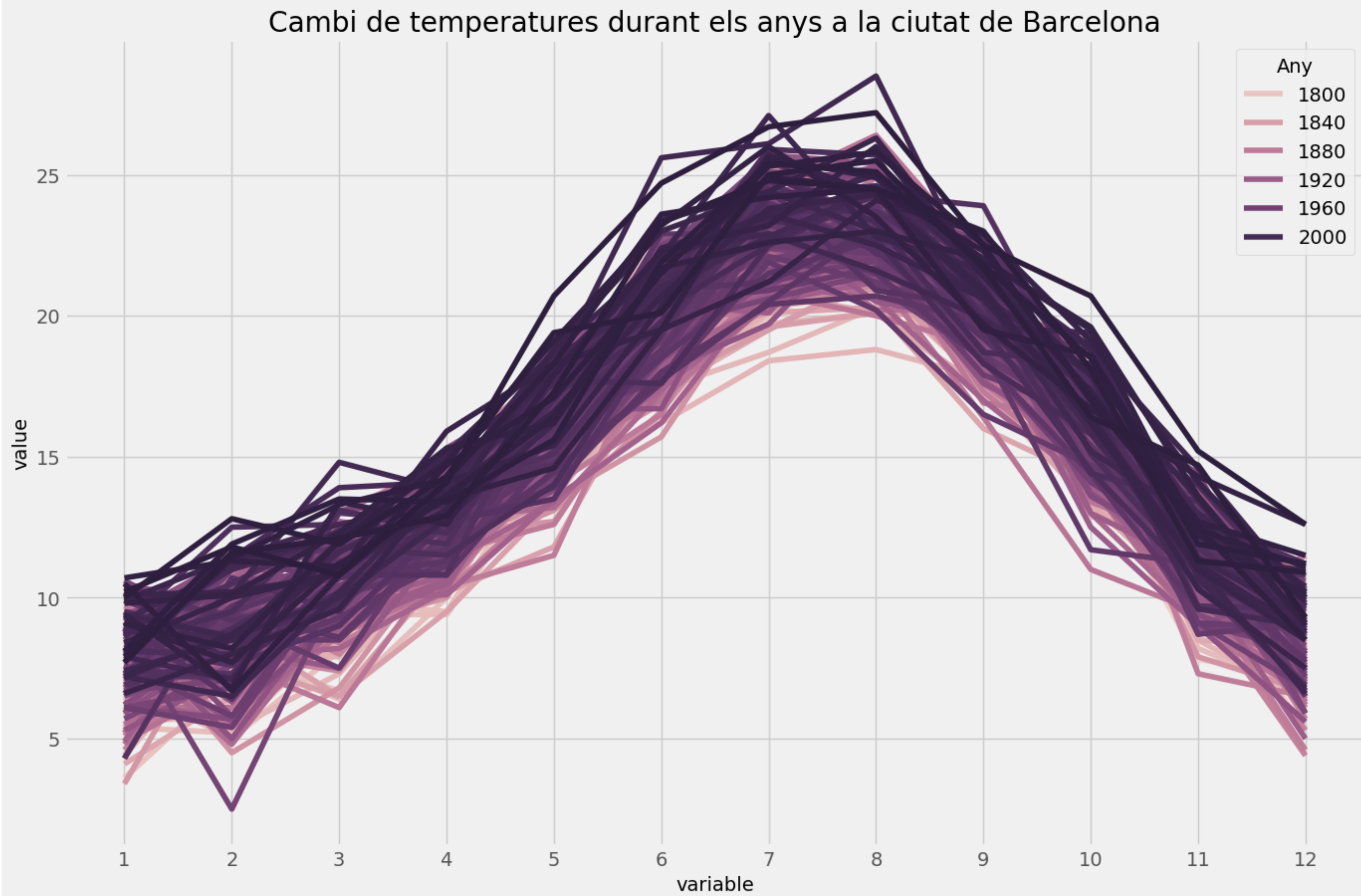
In [13]:
```
table["Data"] = pd.to_datetime(dict(year=table["Any"], month=table["variable"], day="1"))
```

In [14]:
```
table.head()
```

Out[14]:

	Any	variable	value	Data
0	1786	1	7.8	1786-01-01
1	1787	1	5.4	1787-01-01
2	1788	1	6.4	1788-01-01
3	1789	1	6.9	1789-01-01
4	1790	1	7.4	1790-01-01

In [15]:
```
plt.style.use('fivethirtyeight')
plt.figure(figsize=(15, 10))
plt.title("Cambi de temperatures durant els anys a la ciutat de Barcelona")
sns.lineplot(data = table, x='variable', y='value', hue='Any')
plt.show()
```



In [16]:
```
df = table.set_index("Data")
df
```

Out[16]:

Data	Any	variable	value
1786-01-01	1786	1	7.8
1787-01-01	1787	1	5.4
1788-01-01	1788	1	6.4
1789-01-01	1789	1	6.9
1790-01-01	1790	1	7.4
...	...	...	...
2018-12-01	2018	12	11.1
2019-12-01	2019	12	11.2
2020-12-01	2020	12	9.3
2021-12-01	2021	12	10.9
2022-12-01	2022	12	12.6

2844 rows × 3 columns

In [17]:
```
df = df.drop(['Any', 'variable'], axis=1)
```

In [18]:
```
df = df.groupby(['Data']).mean()
```

In [19]:
```
fig = px.line(df, x= df.index, y="value", template = "plotly_dark", title="Temperaturas
```

```
fig.show()
```

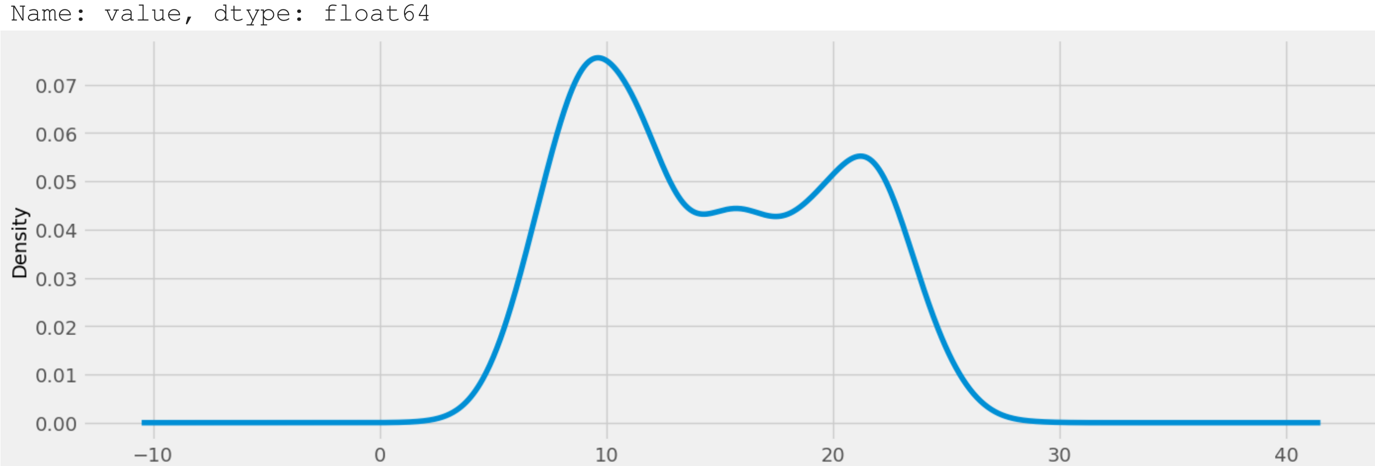
In [20]:
```
def Prueba_Dickey_Fuller(series , column_name):
    print (f'Resultados de la prueba de Dickey-Fuller para columna: {column_name}')
    dftest = adfuller(series, autolag='AIC')
    dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','No Lags Used','
    for key,value in dftest[4].items():
       dfoutput['Critical Value (%s)'%key] = value
    print (dfoutput)
    if dftest[1] <= 0.05:
        print("Conclusion:====>")
        print("Rechazar la hipótesis nula")
        print("Los datos son estacionarios")
    else:
        print("Conclusion:====>")
        print("No se puede rechazar la hipótesis nula")
        print("Los datos no son estacionarios")
```

In [21]:
```
Prueba_Dickey_Fuller(df["value"],"value")
```

```
Resultados de la prueba de Dickey-Fuller para columna: value
Test Statistic                          -3.180457
p-value                                  0.021144
No Lags Used                            25.000000
Número de observaciones utilizadas    2818.000000
Critical Value (1%)                     -3.432673
Critical Value (5%)                     -2.862566
Critical Value (10%)                    -2.567316
dtype: float64
Conclusion:====>
Rechazar la hipótesis nula
Los datos son estacionarios
```

In [22]:
```
df["value"].plot(kind='kde', figsize =(15,5))
df["value"].describe()
```

Out[22]:
```
count    2844.000000
mean       14.593530
std         5.529662
min         2.500000
25%         9.700000
50%        14.000000
75%        19.700000
max        28.500000
Name: value, dtype: float64
```



In [23]:
```
df1=df.copy()
```

In [24]:
```
df1['value_diff'] = df['value'].diff().fillna(0)
df1['value_diff2'] = df1['value_diff'].diff().fillna(0)
```

In [25]:
```
# Take a look at the head of the dataset
df1.head()
```

Out[25]:

Data	value	value_diff	value_diff2
1786-01-01	7.8	0.0	0.0
1786-02-01	8.3	0.5	0.5
1786-03-01	9.9	1.6	1.1
1786-04-01	12.8	2.9	1.3
1786-05-01	16.8	4.0	1.1

In [26]:
```
# df1 = df1.drop(['Any', 'variable'], axis=1)
```

In [27]:
```
Prueba_Dickey_Fuller(df1["value_diff"],"value_diff")
```

```
Resultados de la prueba de Dickey-Fuller para columna: value_diff
Test Statistic                           -22.468216
p-value                                    0.000000
No Lags Used                              23.000000
Número de observaciones utilizadas      2820.000000
Critical Value (1%)                       -3.432671
Critical Value (5%)                       -2.862565
Critical Value (10%)                      -2.567316
dtype: float64
Conclusion:====>
Rechazar la hipótesis nula
Los datos son estacionarios
```

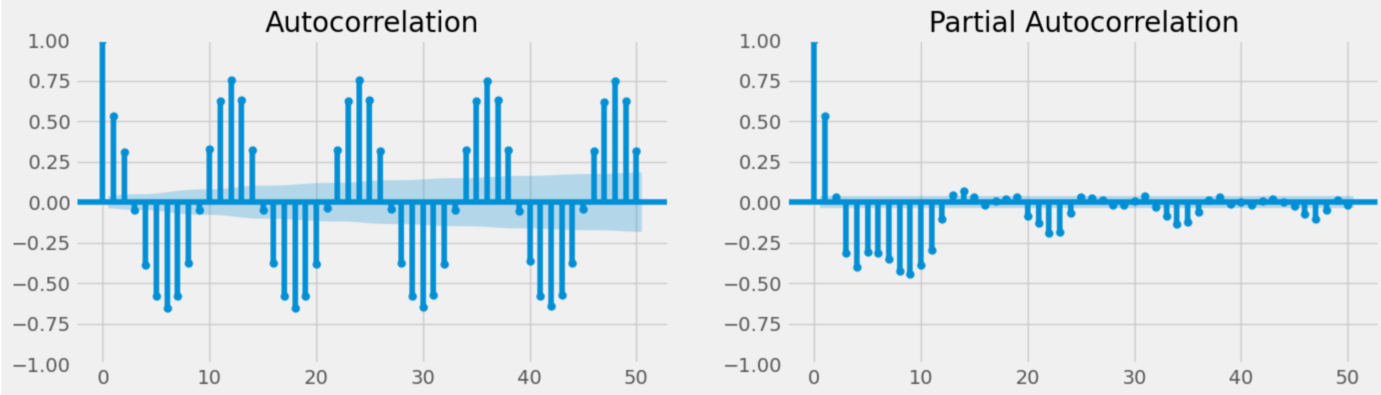
Seasonal ARIMA, es una extensión de ARIMA que admite explícitamente datos de series temporales univariadas con un componente estacional. Agrega tres nuevos hiperparámetros para especificar la autorregresión (AR), diferenciación (I) y media móvil (MA) para el componente estacional de la serie, así como un parámetro adicional para el período de la estacionalidad.

Hay cuatro elementos estacionales que no forman parte de ARIMA que deben configurarse; ellos son:

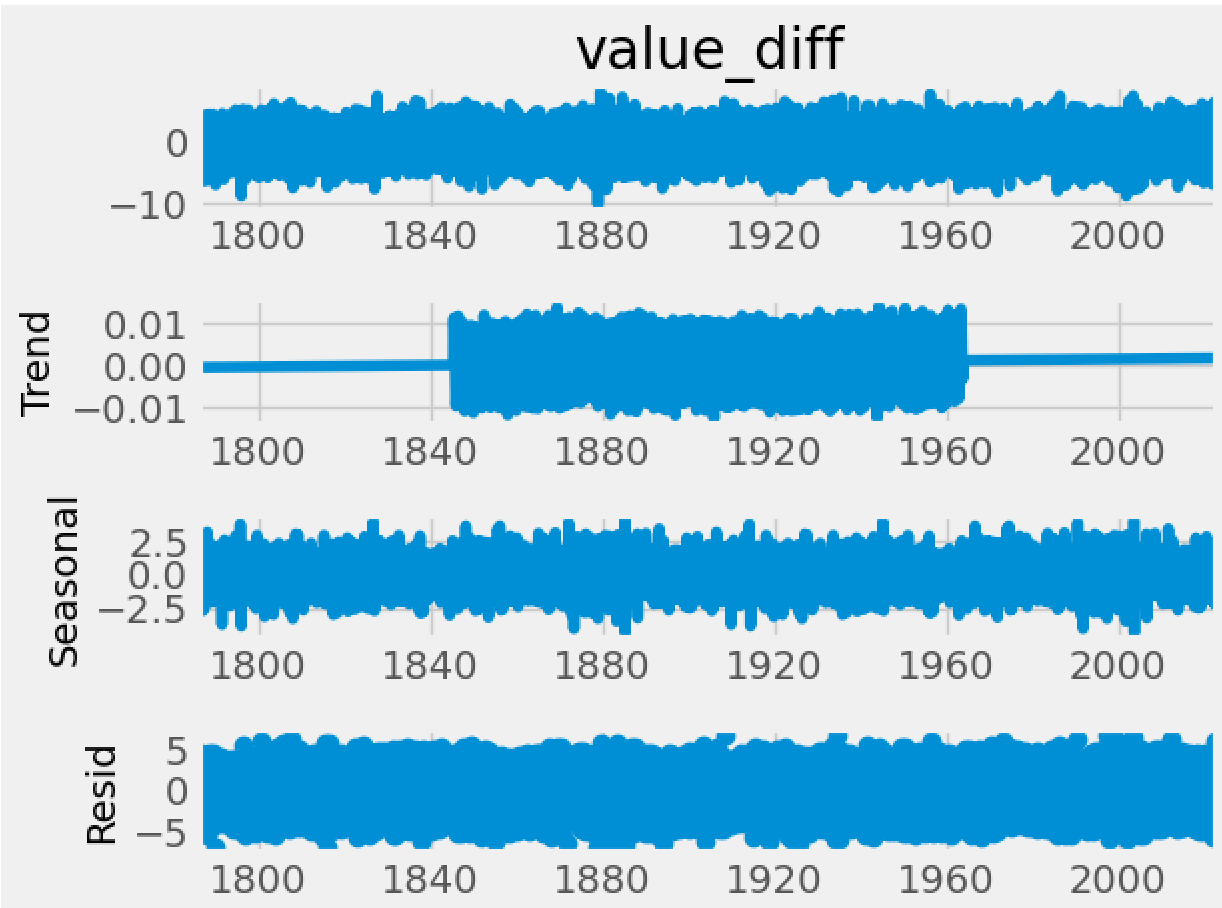
P: orden autorregresivo estacional. D: Orden de diferencia estacional. P: Orden promedio móvil estacional. m: El número de pasos de tiempo para un solo período estacional.

In [28]:
```
fig = px.line(df1, x= df1.index, y="value_diff", template = "plotly_dark", title="Temper
fig.show()
```

In [29]:
```
fig, axes = plt.subplots(1,2,figsize=(15,4))
a = plot_acf( df1["value_diff"],lags=50, ax=axes[0])
b = plot_pacf(df1["value_diff"],lags=50, ax=axes[1])
plt.show(a)
plt.show(b)
```



In [30]:
```
result = seasonal_decompose(df1['value_diff'], model='add',extrapolate_trend='freq', per
result.plot()
plt.show()
```



In [31]:
```
train_data = df[:len(df)-12]
test_data = df[len(df)-12:]
test=test_data.copy()
train_data.shape, test_data.shape
```

Out[31]:
```
((2832, 1), (12, 1))
```

In [32]:
```
print(f"Fechas datos_train : {train_data.index.min()} --- {train_data.index.max()}  (n={

print(f"Fechas datos_test  : {test_data.index.min()} --- {test_data.index.max()}  (n={le
```

```
Fechas datos_train : 1786-01-01 00:00:00 --- 2021-12-01 00:00:00  (n=2832)
Fechas datos_test  : 2022-01-01 00:00:00 --- 2022-12-01 00:00:00  (n=12)
```

In [33]:
```
test_data.head(3)
```

Out[33]:

value

Data	
2022-01-01	10.2
2022-02-01	11.8
2022-03-01	10.8

3.1 AUTOARIMA

In [34]:
```
!pip install pmdarima
```

```
Requirement already satisfied: pmdarima in c:\users\taida\anaconda3\lib\site-packages
(2.0.3)
Requirement already satisfied: joblib>=0.11 in c:\users\taida\anaconda3\lib\site-package
s (from pmdarima) (1.1.1)
Requirement already satisfied: Cython!=0.29.18,!=0.29.31,>=0.29 in c:\users\taida\anacon
da3\lib\site-packages (from pmdarima) (0.29.35)
Requirement already satisfied: numpy>=1.21.2 in c:\users\taida\anaconda3\lib\site-packag
es (from pmdarima) (1.23.5)
Requirement already satisfied: pandas>=0.19 in c:\users\taida\anaconda3\lib\site-package
s (from pmdarima) (1.5.3)
Requirement already satisfied: scikit-learn>=0.22 in c:\users\taida\anaconda3\lib\site-p
ackages (from pmdarima) (1.2.1)
Requirement already satisfied: scipy>=1.3.2 in c:\users\taida\anaconda3\lib\site-package
s (from pmdarima) (1.10.0)
Requirement already satisfied: statsmodels>=0.13.2 in c:\users\taida\anaconda3\lib\site-
packages (from pmdarima) (0.13.5)
Requirement already satisfied: urllib3 in c:\users\taida\anaconda3\lib\site-packages (fr
om pmdarima) (1.26.14)
Requirement already satisfied: setuptools!=50.0.0,>=38.6.0 in c:\users\taida\anaconda3\l
ib\site-packages (from pmdarima) (65.6.3)
Requirement already satisfied: python-dateutil>=2.8.1 in c:\users\taida\anaconda3\lib\si
te-packages (from pandas>=0.19->pmdarima) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\taida\anaconda3\lib\site-package
s (from pandas>=0.19->pmdarima) (2022.7)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\taida\anaconda3\lib\site
-packages (from scikit-learn>=0.22->pmdarima) (2.2.0)
Requirement already satisfied: patsy>=0.5.2 in c:\users\taida\anaconda3\lib\site-package
s (from statsmodels>=0.13.2->pmdarima) (0.5.3)
Requirement already satisfied: packaging>=21.3 in c:\users\taida\anaconda3\lib\site-pack
ages (from statsmodels>=0.13.2->pmdarima) (22.0)
Requirement already satisfied: six in c:\users\taida\anaconda3\lib\site-packages (from p
atsy>=0.5.2->statsmodels>=0.13.2->pmdarima) (1.16.0)
```

In [35]:
```
from pmdarima import auto_arima
```

In [36]:
```
modelo_auto=auto_arima(train_data,start_p=0,d=1,start_q=0,
          max_p=4,max_d=2,max_q=4, start_P=0,
          D=1, start_Q=0, max_P=2,max_D=1,
          max_Q=2, m=12, seasonal=True,
          error_action='warn',trace=True,
          supress_warnings=True,stepwise=True,
          random_state=20,n_fits=50)
print(modelo_auto)
```

```
Performing stepwise search to minimize aic
 ARIMA(0,1,0)(0,1,0)[12]             : AIC=12627.535, Time=0.32 sec
 ARIMA(1,1,0)(1,1,0)[12]             : AIC=11285.044, Time=0.73 sec
 ARIMA(0,1,1)(0,1,1)[12]             : AIC=inf, Time=2.07 sec
 ARIMA(1,1,0)(0,1,0)[12]             : AIC=12124.025, Time=0.26 sec
 ARIMA(1,1,0)(2,1,0)[12]             : AIC=10968.007, Time=1.51 sec
 ARIMA(1,1,0)(2,1,1)[12]             : AIC=inf, Time=5.63 sec
 ARIMA(1,1,0)(1,1,1)[12]             : AIC=inf, Time=3.10 sec
 ARIMA(0,1,0)(2,1,0)[12]             : AIC=11484.919, Time=1.18 sec
```

```
 ARIMA(2,1,0)(2,1,0)[12]             : AIC=10724.424, Time=2.08 sec
 ARIMA(2,1,0)(1,1,0)[12]             : AIC=11055.551, Time=0.78 sec
 ARIMA(2,1,0)(2,1,1)[12]             : AIC=inf, Time=7.44 sec
 ARIMA(2,1,0)(1,1,1)[12]             : AIC=inf, Time=2.54 sec
 ARIMA(3,1,0)(2,1,0)[12]             : AIC=10610.684, Time=1.53 sec
 ARIMA(3,1,0)(1,1,0)[12]             : AIC=10932.769, Time=0.88 sec
 ARIMA(3,1,0)(2,1,1)[12]             : AIC=inf, Time=8.94 sec
 ARIMA(3,1,0)(1,1,1)[12]             : AIC=inf, Time=3.71 sec
 ARIMA(4,1,0)(2,1,0)[12]             : AIC=10492.563, Time=4.94 sec
 ARIMA(4,1,0)(1,1,0)[12]             : AIC=10819.626, Time=2.05 sec
 ARIMA(4,1,0)(2,1,1)[12]             : AIC=inf, Time=12.01 sec
 ARIMA(4,1,0)(1,1,1)[12]             : AIC=inf, Time=3.57 sec
 ARIMA(4,1,1)(2,1,0)[12]             : AIC=inf, Time=19.79 sec
 ARIMA(3,1,1)(2,1,0)[12]             : AIC=inf, Time=13.22 sec
 ARIMA(4,1,0)(2,1,0)[12] intercept   : AIC=10494.563, Time=6.91 sec

Best model:  ARIMA(4,1,0)(2,1,0)[12]
Total fit time: 105.217 seconds
 ARIMA(4,1,0)(2,1,0)[12]
```

In [37]:

```python
print(modelo_auto.summary())
```

```
                                      SARIMAX Results                                       
==========================================================================================
==
Dep. Variable:                                  y   No. Observations:                   28
32
Model:             SARIMAX(4, 1, 0)x(2, 1, 0, 12)   Log Likelihood                 -5239.2
82
Date:                            Mon, 10 Jul 2023   AIC                            10492.5
63
Time:                                    10:38:28   BIC                            10534.1
72
Sample:                                01-01-1786   HQIC                           10507.5
77
                                     - 12-01-2021                                         
Covariance Type:                              opg                                         
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.6277      0.018    -35.106      0.000      -0.663      -0.593
ar.L2         -0.4755      0.021    -22.668      0.000      -0.517      -0.434
ar.L3         -0.3205      0.021    -15.239      0.000      -0.362      -0.279
ar.L4         -0.2043      0.018    -11.215      0.000      -0.240      -0.169
ar.S.L12      -0.6779      0.017    -40.091      0.000      -0.711      -0.645
ar.S.L24      -0.3335      0.017    -19.160      0.000      -0.368      -0.299
sigma2         2.4030      0.059     40.781      0.000       2.288       2.518
===================================================================================
Ljung-Box (L1) (Q):                   2.77   Jarque-Bera (JB):                19.00
Prob(Q):                              0.10   Prob(JB):                         0.00
Heteroskedasticity (H):               1.20   Skew:                            -0.07
Prob(H) (two-sided):                  0.01   Kurtosis:                         3.37
===================================================================================

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

In [38]:

```python
arima_model = SARIMAX(train_data["value"], order = (4,1,0), seasonal_order = (2,1,0,12))
arima_result = arima_model.fit()
arima_result.summary()
```

Out[38]:

SARIMAX Results

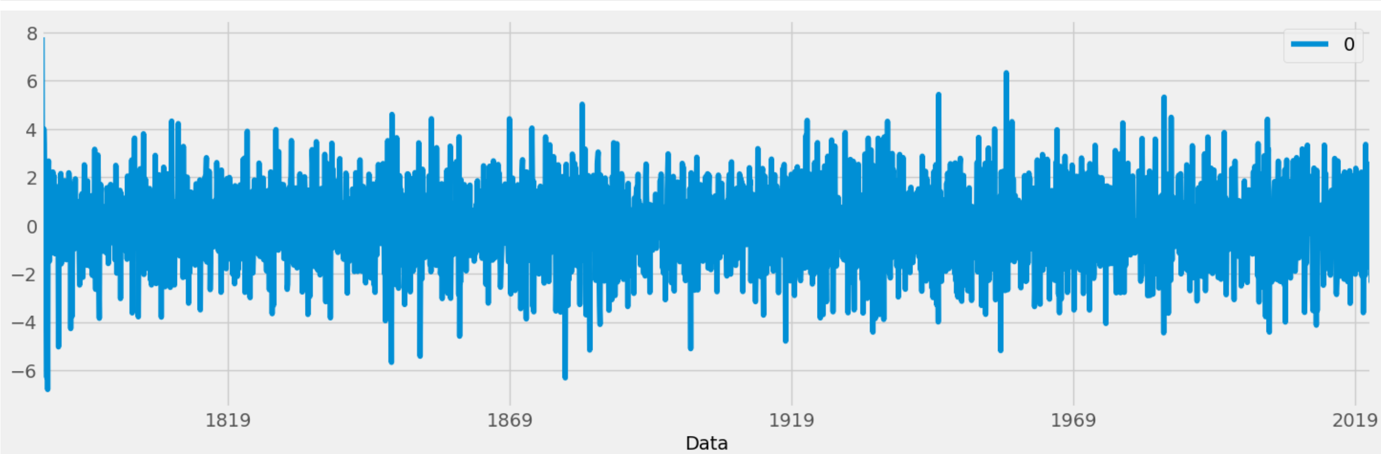
Dep. Variable:	value	**No. Observations:**	2832
Model:	SARIMAX(4, 1, 0)x(2, 1, 0, 12)	**Log Likelihood**	-5239.282
Date:	Mon, 10 Jul 2023	**AIC**	10492.563
Time:	10:38:30	**BIC**	10534.172
Sample:	01-01-1786	**HQIC**	10507.577
	- 12-01-2021		
Covariance Type:	opg		

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
ar.L1	-0.6277	0.018	-35.106	0.000	-0.663	-0.593
ar.L2	-0.4755	0.021	-22.668	0.000	-0.517	-0.434
ar.L3	-0.3205	0.021	-15.239	0.000	-0.362	-0.279
ar.L4	-0.2043	0.018	-11.215	0.000	-0.240	-0.169
ar.S.L12	-0.6779	0.017	-40.091	0.000	-0.711	-0.645
ar.S.L24	-0.3335	0.017	-19.160	0.000	-0.368	-0.299
sigma2	2.4030	0.059	40.781	0.000	2.288	2.518

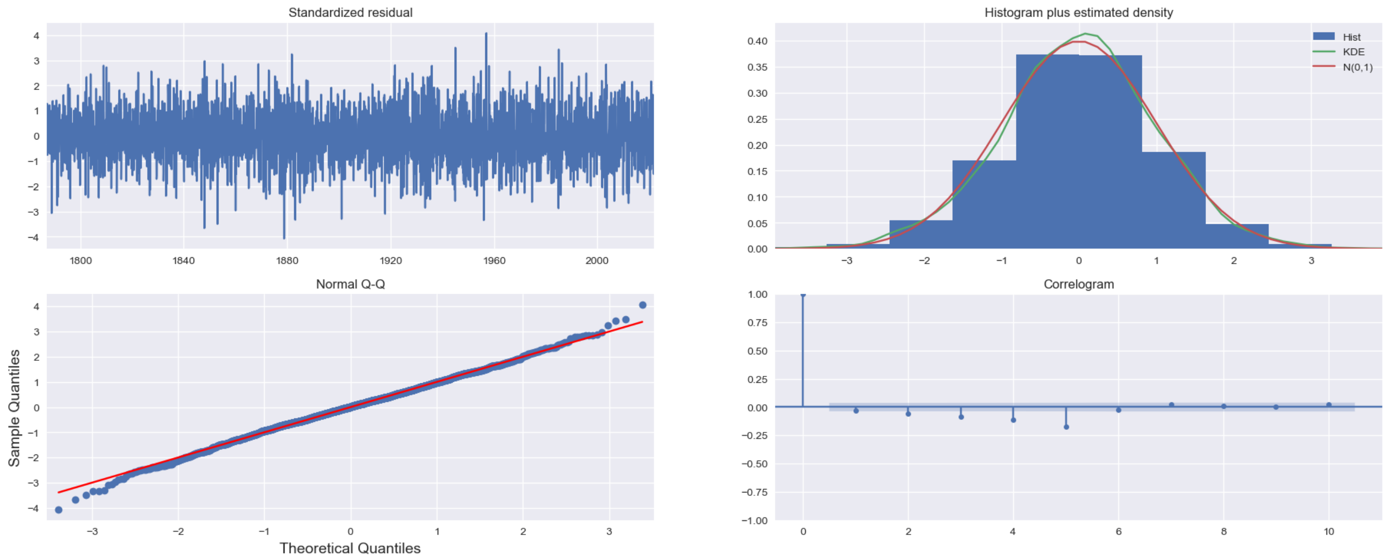
Ljung-Box (L1) (Q):	2.77	**Jarque-Bera (JB):**	19.00
Prob(Q):	0.10	**Prob(JB):**	0.00
Heteroskedasticity (H):	1.20	**Skew:**	-0.07
Prob(H) (two-sided):	0.01	**Kurtosis:**	3.37

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

In [39]:
```
# Gráfico de línea de errores residuales
residuals = pd.DataFrame(arima_result.resid)
residuals.plot(figsize = (16,5));
plt.show();
```



In [40]:
```
plt.style.use('seaborn')
modelo_auto.plot_diagnostics(figsize=(20,8))
plt.show()
```

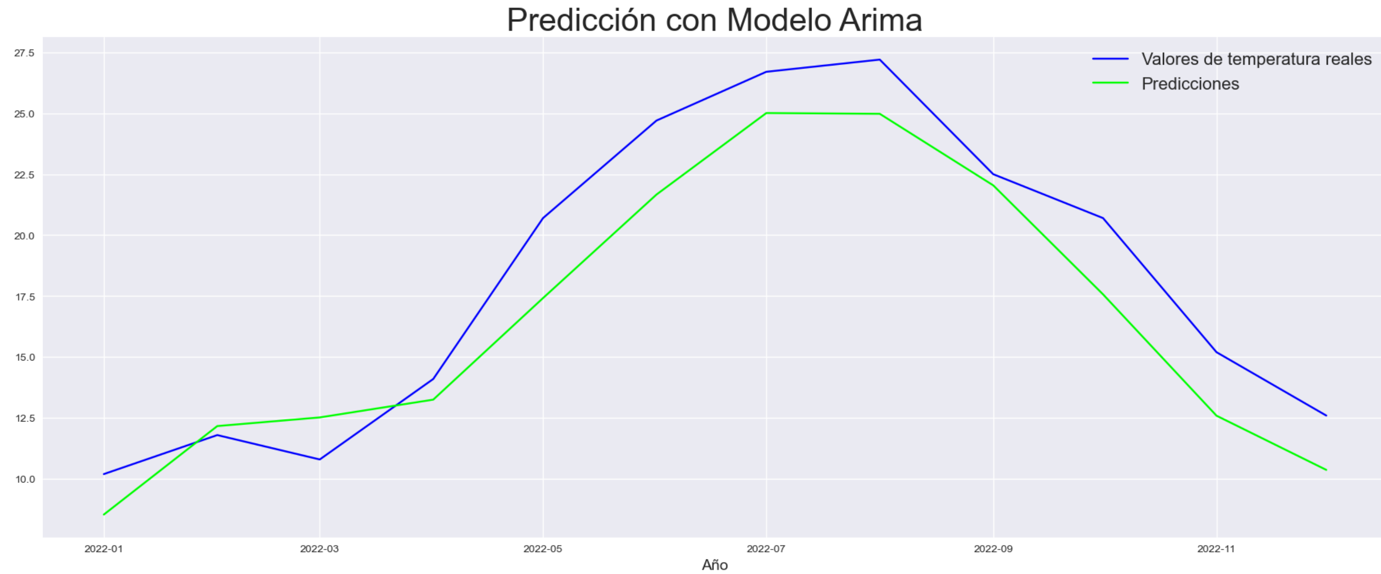


In [41]:
```
len(train_data)
```
Out[41]:
```
2832
```

In [42]:
```
len(df)-1
```
Out[42]:
```
2843
```

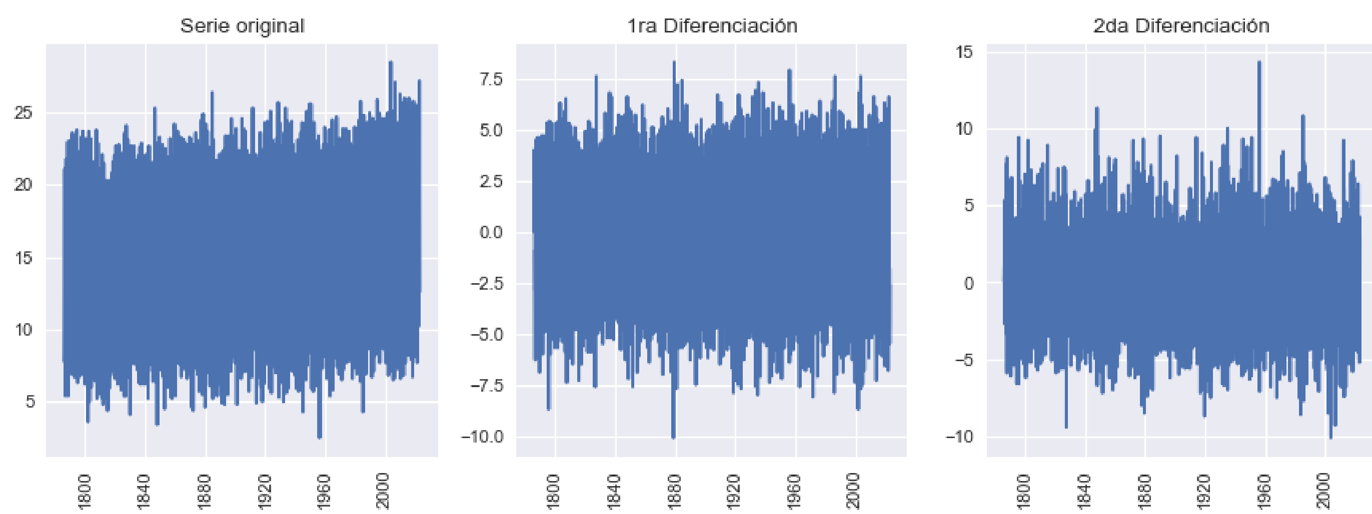
In [43]:
```
arima_pred = arima_result.predict(start = len(train_data), end = len(df)-1, typ="levels"
arima_pred
```
Out[43]:
```
2022-01-01     8.540187
2022-02-01    12.169738
2022-03-01    12.524241
2022-04-01    13.253313
2022-05-01    17.419700
2022-06-01    21.665486
2022-07-01    25.007749
2022-08-01    24.977080
2022-09-01    22.042841
2022-10-01    17.572318
2022-11-01    12.595473
2022-12-01    10.371998
Freq: MS, Name: ARIMA Predictions, dtype: float64
```

In [44]:
```
plt.style.use('seaborn')
plt.rcParams["figure.figsize"] = (20, 8)

plt.plot(test_data["value"],color="blue" ,label="Valores de temperatura reales")
plt.plot(arima_pred, color="lime", label="Predicciones")
plt.title("Predicción con Modelo Arima", fontsize=30);
plt.xlabel('Año')
plt.ylabel('')
plt.legend( fontsize=16);
plt.show();
```



In [45]:

```
#ploting the original and diferenced series.

plt.rcParams.update({'figure.figsize':(12,4), 'figure.dpi':80})

# Original Series
fig, axes = plt.subplots(1, 3, sharex=True)
axes[0].plot(df['value'])
axes[0].set_title('Serie original')

# 1st Diff
axes[1].plot(df1['value_diff'])
axes[1].set_title('1ra Diferenciación')

# 2th Diff
axes[2].plot(df1['value_diff2'])
axes[2].set_title('2da Diferenciación')

for ax in fig.axes:
    plt.sca(ax)
    plt.xticks(rotation=90)
plt.show()
```



In [46]:

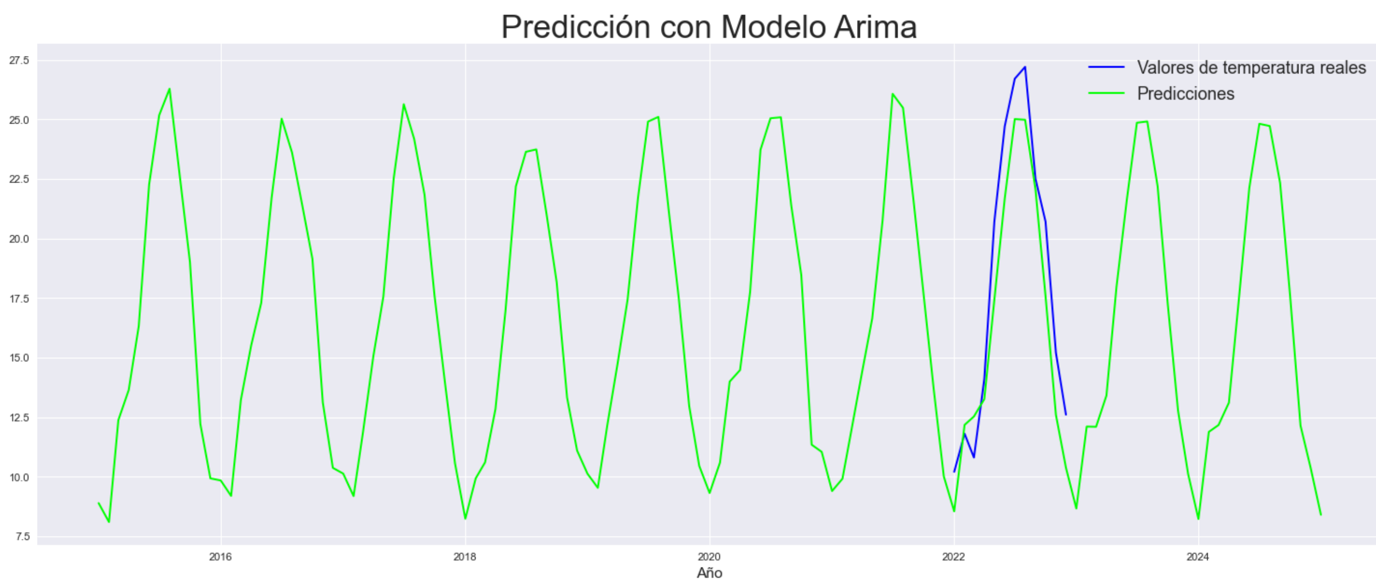
```
arima_pred2 = arima_result.predict(start='2015-01-01',end='2025-01-01', typ="levels").re
arima_pred2
```

Out[46]:

```
2015-01-01     8.888517
2015-02-01     8.093180
2015-03-01    12.370532
2015-04-01    13.635411
```

```
2015-05-01    16.321257
                 ...
2024-09-01    22.326211
2024-10-01    17.575702
2024-11-01    12.140576
2024-12-01    10.386256
2025-01-01     8.398085
Freq: MS, Name: ARIMA Predictions, Length: 121, dtype: float64
```

In [47]:
```python
plt.style.use('seaborn')
plt.rcParams["figure.figsize"] = (20, 8)

plt.plot(test_data["value"],color="blue" ,label="Valores de temperatura reales")
plt.plot(arima_pred2, color="lime", label="Predicciones")
plt.title("Predicción con Modelo Arima", fontsize=30);
plt.xlabel('Año')
plt.ylabel('')
plt.legend( fontsize=16);
plt.show();
```



In [48]:
```python
def evaluacion_metrica(y_true, y_pred):

    def mean_absolute_percentage_error(y_true, y_pred):
        y_true, y_pred = np.array(y_true), np.array(y_pred)
        return np.mean(np.abs((y_true - y_pred) / y_true)) * 100
    print('Evaluation metric results:-')
    print(f'MSE is : {metrics.mean_squared_error(y_true, y_pred)}')
    print(f'MAE is : {metrics.mean_absolute_error(y_true, y_pred)}')
    print(f'RMSE is : {np.sqrt(metrics.mean_squared_error(y_true, y_pred))}')
    print(f'MAPE is : {mean_absolute_percentage_error(y_true, y_pred)}')
    print(f'R2 is : {metrics.r2_score(y_true, y_pred)}',end='\n\n')
```

In [49]:
```python
evaluacion_metrica(test_data["value"],arima_pred)
```

```
Evaluation metric results:-
MSE is : 4.674520995909241
MAE is : 1.9373197066149803
RMSE is : 2.162064059159497
MAPE is : 11.331498672440492
R2 is : 0.8734506791704397
```

In [108…
```python
test_data['ARIMA_Predictions']= arima_pred
test_data
```

Out[108]:

	value	LSTM_Predictions	Prophet_Predictions	ARIMA_Predictions

Data				
2022-01-01	10.2	8.241807	12.283587	8.540187
2022-02-01	11.8	9.503069	12.139533	12.169738
2022-03-01	10.8	10.972598	11.985402	12.524241
2022-04-01	14.1	13.185527	11.823201	13.253313
2022-05-01	20.7	16.656366	11.655270	17.419700
2022-06-01	24.7	21.280418	11.484218	21.665486
2022-07-01	26.7	24.313978	11.312851	25.007749
2022-08-01	27.2	24.098545	11.144093	24.977080
2022-09-01	22.5	21.644652	10.980907	22.042841
2022-10-01	20.7	17.169369	10.826205	17.572318
2022-11-01	15.2	11.872340	10.682767	12.595473
2022-12-01	12.6	8.568330	10.553160	10.371998

3.2 LSTM Forecast LSTM significa memoria a corto plazo. Es un modelo o arquitectura que amplía la memoria de las redes neuronales recurrentes. Por lo general, las redes neuronales recurrentes tienen "memoria a corto plazo" en el sentido de que utilizan información anterior persistente para ser utilizada en la red neuronal actual. Esencialmente, la información anterior se utiliza en la presente tarea. Eso significa que no tenemos una lista de toda la información anterior disponible para el nodo neuronal. LSTM introduce la memoria a largo plazo en las redes neuronales recurrentes. Mitiga el problema del gradiente de fuga, que es donde la red neuronal deja de aprender porque las actualizaciones de los diversos pesos dentro de una red neuronal dada se vuelven cada vez más pequeñas. Lo hace mediante el uso de una serie de "puertas". Estos están contenidos en bloques de memoria que están conectados a través de capas, así:

In [51]:
```
# Manipulación y tratamiento de Datos
import numpy as np
import pandas as pd

# Visualización de datos
import seaborn as sns
import plotly.express as px
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('ggplot')

import tensorflow as tf
from keras.preprocessing.sequence import TimeseriesGenerator
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from tensorflow.keras.optimizers.legacy import Adam

# Métrica de Evaluación
from sklearn.metrics import mean_squared_error
from statsmodels.tools.eval_measures import rmse
from sklearn import metrics

# No presentar advertencias
import warnings
warnings.filterwarnings("ignore")
```

In [52]:
```
#Estandarización
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
```

In [55]:
```
df
```

Out[55]:

	value
Data	
1786-01-01	7.8
1786-02-01	8.3
1786-03-01	9.9
1786-04-01	12.8
1786-05-01	16.8
...	...
2022-08-01	27.2
2022-09-01	22.5
2022-10-01	20.7
2022-11-01	15.2
2022-12-01	12.6

2844 rows × 1 columns

In [56]:
```
train_data = df[:len(df)-12]
test_data = df[len(df)-12:]
test=test_data.copy()
train_data.shape, test_data.shape
```

Out[56]:
```
((2832, 1), (12, 1))
```

In [57]:
```
scaler.fit(train_data)
scaled_train_data = scaler.transform(train_data)

scaled_test_data = scaler.transform(test)
```

In [58]:
```
n_input = 12
n_features= 1
generator = TimeseriesGenerator(scaled_train_data, scaled_train_data, length=n_input, ba
```

In [59]:
```
lstm_model = Sequential()
lstm_model.add(LSTM(200, activation='relu', input_shape=(n_input, n_features)))
lstm_model.add(Dense(1))
lstm_model.compile(optimizer='adam', loss='mse')

lstm_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 200)               161600

 dense (Dense)               (None, 1)                 201

=================================================================
```

```
Total params: 161,801
Trainable params: 161,801
Non-trainable params: 0
_________________________________________________________________
```

In [60]:

```python
lstm_model.fit_generator(generator,epochs=50)
```

```
Epoch 1/50
2820/2820 [==============================] - 13s 4ms/step - loss: 0.0085
Epoch 2/50
2820/2820 [==============================] - 10s 4ms/step - loss: 0.0045
Epoch 3/50
2820/2820 [==============================] - 11s 4ms/step - loss: 0.0037
Epoch 4/50
2820/2820 [==============================] - 12s 4ms/step - loss: 0.0035
Epoch 5/50
2820/2820 [==============================] - 12s 4ms/step - loss: 0.0034
Epoch 6/50
2820/2820 [==============================] - 12s 4ms/step - loss: 0.0032
Epoch 7/50
2820/2820 [==============================] - 12s 4ms/step - loss: 0.0031
Epoch 8/50
2820/2820 [==============================] - 11s 4ms/step - loss: 0.0032
Epoch 9/50
2820/2820 [==============================] - 11s 4ms/step - loss: 0.0031
Epoch 10/50
2820/2820 [==============================] - 12s 4ms/step - loss: 0.0031
Epoch 11/50
2820/2820 [==============================] - 12s 4ms/step - loss: 0.0030
Epoch 12/50
2820/2820 [==============================] - 10s 4ms/step - loss: 0.0030
Epoch 13/50
2820/2820 [==============================] - 10s 4ms/step - loss: 0.0030
Epoch 14/50
2820/2820 [==============================] - 10s 4ms/step - loss: 0.0030
Epoch 15/50
2820/2820 [==============================] - 10s 3ms/step - loss: 0.0030
Epoch 16/50
2820/2820 [==============================] - 11s 4ms/step - loss: 0.0029
Epoch 17/50
2820/2820 [==============================] - 11s 4ms/step - loss: 0.0029
Epoch 18/50
2820/2820 [==============================] - 11s 4ms/step - loss: 0.0029
Epoch 19/50
2820/2820 [==============================] - 11s 4ms/step - loss: 0.0029
Epoch 20/50
2820/2820 [==============================] - 11s 4ms/step - loss: 0.0029
Epoch 21/50
2820/2820 [==============================] - 11s 4ms/step - loss: 0.0029
Epoch 22/50
2820/2820 [==============================] - 11s 4ms/step - loss: 0.0029
Epoch 23/50
2820/2820 [==============================] - 12s 4ms/step - loss: 0.0029
Epoch 24/50
2820/2820 [==============================] - 11s 4ms/step - loss: 0.0029
Epoch 25/50
2820/2820 [==============================] - 11s 4ms/step - loss: 0.0029
Epoch 26/50
2820/2820 [==============================] - 11s 4ms/step - loss: 0.0029
Epoch 27/50
2820/2820 [==============================] - 11s 4ms/step - loss: 0.0029
Epoch 28/50
2820/2820 [==============================] - 11s 4ms/step - loss: 0.0029
Epoch 29/50
2820/2820 [==============================] - 11s 4ms/step - loss: 0.0029
Epoch 30/50
```

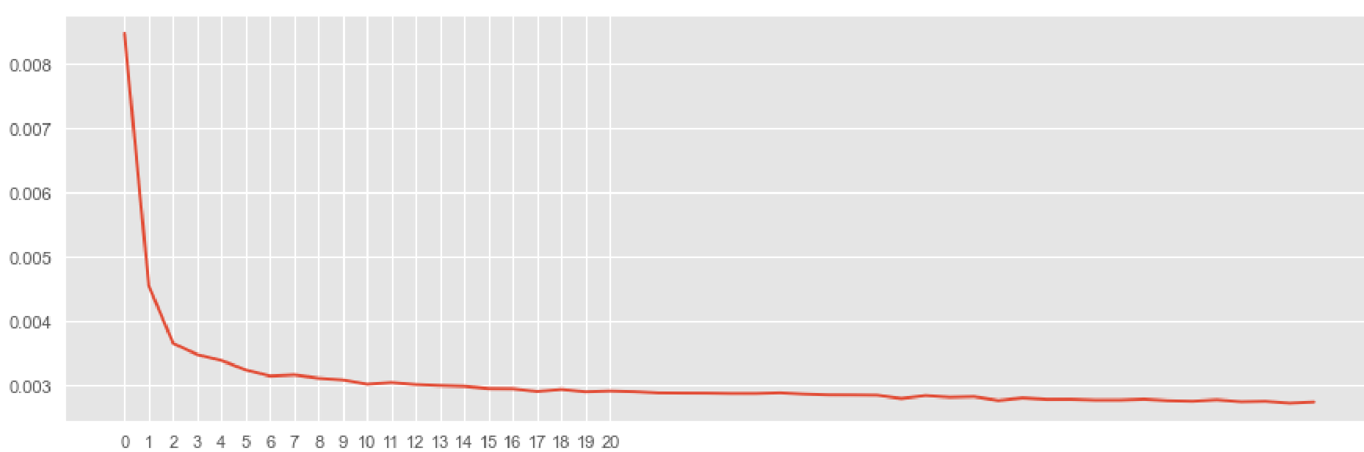
```
2820/2820 [==============================] - 11s 4ms/step - loss: 0.0028
Epoch 31/50
2820/2820 [==============================] - 12s 4ms/step - loss: 0.0028
Epoch 32/50
2820/2820 [==============================] - 12s 4ms/step - loss: 0.0028
Epoch 33/50
2820/2820 [==============================] - 12s 4ms/step - loss: 0.0028
Epoch 34/50
2820/2820 [==============================] - 13s 4ms/step - loss: 0.0028
Epoch 35/50
2820/2820 [==============================] - 13s 5ms/step - loss: 0.0028
Epoch 36/50
2820/2820 [==============================] - 12s 4ms/step - loss: 0.0028
Epoch 37/50
2820/2820 [==============================] - 12s 4ms/step - loss: 0.0028
Epoch 38/50
2820/2820 [==============================] - 12s 4ms/step - loss: 0.0028
Epoch 39/50
2820/2820 [==============================] - 12s 4ms/step - loss: 0.0028
Epoch 40/50
2820/2820 [==============================] - 13s 5ms/step - loss: 0.0028
Epoch 41/50
2820/2820 [==============================] - 11s 4ms/step - loss: 0.0028
Epoch 42/50
2820/2820 [==============================] - 12s 4ms/step - loss: 0.0028
Epoch 43/50
2820/2820 [==============================] - 12s 4ms/step - loss: 0.0028
Epoch 44/50
2820/2820 [==============================] - 12s 4ms/step - loss: 0.0028
Epoch 45/50
2820/2820 [==============================] - 12s 4ms/step - loss: 0.0028
Epoch 46/50
2820/2820 [==============================] - 11s 4ms/step - loss: 0.0028
Epoch 47/50
2820/2820 [==============================] - 11s 4ms/step - loss: 0.0027
Epoch 48/50
2820/2820 [==============================] - 11s 4ms/step - loss: 0.0028
Epoch 49/50
2820/2820 [==============================] - 11s 4ms/step - loss: 0.0027
Epoch 50/50
2820/2820 [==============================] - 11s 4ms/step - loss: 0.0027
```

Out[60]:

```
<keras.callbacks.History at 0x2b09d9adf60>
```

In [61]:

```python
losses_lstm = lstm_model.history.history['loss']
plt.figure(figsize=(12,4))
plt.xticks(np.arange(0,21,1))
plt.plot(range(len(losses_lstm)),losses_lstm);
```



In [62]:

```python
lstm_predictions_scaled = list()
```

```
batch = scaled_train_data[-n_input:]
current_batch = batch.reshape((1, n_input, n_features))

for i in range(len(test_data)):
    lstm_pred = lstm_model.predict(current_batch)[0]
    lstm_predictions_scaled.append(lstm_pred)
    current_batch = np.append(current_batch[:,1:,:],[[lstm_pred]],axis=1)
```

```
1/1 [==============================] - 0s 161ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 22ms/step
```

In [63]:
```
lstm_predictions = scaler.inverse_transform(lstm_predictions_scaled)
```

In [64]:
```
lstm_predictions
```

Out[64]:
```
array([[ 8.24180725],
       [ 9.5030688 ],
       [10.97259772],
       [13.18552691],
       [16.65636599],
       [21.28041816],
       [24.31397796],
       [24.09854507],
       [21.64465213],
       [17.16936851],
       [11.87234002],
       [ 8.56832993]])
```

In [65]:
```
test_data['LSTM_Predictions'] = lstm_predictions
```

In [94]:
```
test_data
```

Out[94]:

Data	value	LSTM_Predictions
2022-01-01	10.2	8.241807
2022-02-01	11.8	9.503069
2022-03-01	10.8	10.972598
2022-04-01	14.1	13.185527
2022-05-01	20.7	16.656366
2022-06-01	24.7	21.280418
2022-07-01	26.7	24.313978
2022-08-01	27.2	24.098545
2022-09-01	22.5	21.644652
2022-10-01	20.7	17.169369
2022-11-01	15.2	11.872340

2022-12-01	12.6	8.568330

In [66]:
```
#  template = "plotly_dark"
ai=test_data[["value","LSTM_Predictions"]]
fig = px.line(ai, x=test_data.index, y=ai.columns,title="Predicción con Modelo LSTM")
fig.show()
```

In [67]:
```
evaluacion_metrica(test_data["value"],test_data["LSTM_Predictions"])
```

```
Evaluation metric results:-
MSE is : 7.821480482432121
MAE is : 2.5031830822428063
RMSE is : 2.7966909880128195
MAPE is : 14.601050119011013
R2 is : 0.7882557284907605
```

3.3 Prophet

In [80]:
```
from prophet import Prophet
from prophet.plot import plot_plotly, plot_components_plotly
```

In [82]:
```
table
```

Out[82]:

	Any	variable	value	Data
0	1786	1	7.8	1786-01-01
1	1787	1	5.4	1787-01-01
2	1788	1	6.4	1788-01-01
3	1789	1	6.9	1789-01-01
4	1790	1	7.4	1790-01-01
...	...	...	...	...
2839	2018	12	11.1	2018-12-01
2840	2019	12	11.2	2019-12-01
2841	2020	12	9.3	2020-12-01
2842	2021	12	10.9	2021-12-01
2843	2022	12	12.6	2022-12-01

2844 rows × 4 columns

In [83]:
```
df2 = table.drop(['Any', 'variable'], axis=1)
```

In [84]:
```
df2
```

Out[84]:

	value	Data
0	7.8	1786-01-01
1	5.4	1787-01-01
2	6.4	1788-01-01
3	6.9	1789-01-01

4	7.4	1790-01-01
...	...	...
2839	11.1	2018-12-01
2840	11.2	2019-12-01
2841	9.3	2020-12-01
2842	10.9	2021-12-01
2843	12.6	2022-12-01

2844 rows × 2 columns

In [86]:
```
forecast_data = df2.rename(columns = {"Data": "ds",
                                       "value": "y"})
print(forecast_data)
```

```
         y         ds
0      7.8 1786-01-01
1      5.4 1787-01-01
2      6.4 1788-01-01
3      6.9 1789-01-01
4      7.4 1790-01-01
...    ...        ...
2839  11.1 2018-12-01
2840  11.2 2019-12-01
2841   9.3 2020-12-01
2842  10.9 2021-12-01
2843  12.6 2022-12-01

[2844 rows x 2 columns]
```

In [110…
```
model = Prophet()
model.fit(forecast_data)
forecasts = model.make_future_dataframe(periods=0)
predictions = model.predict(forecasts)
plot_plotly(model, predictions)
```

```
11:06:06 - cmdstanpy - INFO - Chain [1] start processing
11:06:06 - cmdstanpy - INFO - Chain [1] done processing
```

In [111…
```
predictions
```

Out[111]:

	ds	trend	yhat_lower	yhat_upper	trend_lower	trend_upper	additive_terms	additive_terms_lower
0	1786-01-01	14.344762	5.176709	8.649631	14.344762	14.344762	-7.283081	-7.283081
1	1786-02-01	14.342961	7.069633	10.523925	14.342961	14.342961	-5.566759	-5.566759
2	1786-03-01	14.341334	8.293690	11.584400	14.341334	14.341334	-4.456900	-4.456900
3	1786-04-01	14.339533	10.861576	14.099918	14.339533	14.339533	-1.804321	-1.804321
4	1786-05-01	14.337790	14.437469	17.736440	14.337790	14.337790	1.807263	1.807263
...	...	...	...	...	...	...	...	...
2839	2022-08-01	16.637867	22.853655	26.273183	16.637867	16.637867	8.009990	8.009990

2840	2022-09-01	16.641300	20.391915	23.750015	16.641300	16.641300	5.298160	5.298160
2841	2022-10-01	16.644622	16.143874	19.597731	16.644622	16.644622	1.304702	1.304702
2842	2022-11-01	16.648055	11.641202	14.963608	16.648055	16.648055	-3.327048	-3.327048
2843	2022-12-01	16.651377	8.770048	12.155413	16.651377	16.651377	-6.168629	-6.168629

2844 rows × 16 columns

In [112…

```
prophet_pred = pd.DataFrame({"Date" : predictions[-12:]['ds'], "Pred" : predictions[-12:
```

In [113…

```
prophet_pred = prophet_pred.set_index("Date")
```

In [114…

```
prophet_pred
```

Out[114]:

Date	Pred
2022-01-01	9.787926
2022-02-01	10.528691
2022-03-01	12.420868
2022-04-01	14.360580
2022-05-01	17.729398
2022-06-01	21.737026
2022-07-01	24.642527
2022-08-01	24.647857
2022-09-01	21.939460
2022-10-01	17.949324
2022-11-01	13.321006
2022-12-01	10.482748

In [115…

```
test_data["Prophet_Predictions"] = prophet_pred['Pred'].values
```

In [116…

```
test_data.head()
```

Out[116]:

Data	value	LSTM_Predictions	Prophet_Predictions	ARIMA_Predictions
2022-01-01	10.2	8.241807	9.787926	8.540187
2022-02-01	11.8	9.503069	10.528691	12.169738
2022-03-01	10.8	10.972598	12.420868	12.524241
2022-04-01	14.1	13.185527	14.360580	13.253313
2022-05-01	20.7	16.656366	17.729398	17.419700

In [120…

```
ai2=test_data[["value","Prophet_Predictions"]]
fig = px.line(ai2, x=test_data.index, y=ai2.columns,title="Predicción con Modelo Prophet
fig.show()
```

In [117…

```
evaluacion_metrica(test_data["value"],test_data["Prophet_Predictions"])
```

```
Evaluation metric results:-
MSE is : 4.06043855684281
MAE is : 1.7846236948827119
RMSE is : 2.015052991075622
MAPE is : 10.004181981951
R2 is : 0.8900752093982905
```

In [118…

```
plt.figure(figsize=(16,9))
plt.plot_date(test_data.index, test_data["value"],label="Original", linestyle="-")
plt.plot_date(test_data.index, test_data["ARIMA_Predictions"], label="Arima",linestyle="
plt.plot_date(test_data.index, test_data["LSTM_Predictions"],label="LSTM", linestyle="--
plt.plot_date(test_data.index, test_data["Prophet_Predictions"], label="Prophet",linesty
plt.legend(fontsize=12)
plt.title("Prediciones de los Diferentes Modelos", fontsize=22)
plt.show();
```