Time Series -Temperature

October 20, 2023

1 Northern Hemisphere Monthly Temperature 1880-2022

As we know, the global warming is real, and it is happening. The temperature increases each year, affecting the life of all the planet.

This data set contains the temperature of the Northern Hemisphere Monthly from 1880 to 2022 (Combined Land-Surface Air and Sea-Surface Water Temperature Anomalies), and I want to analyze how it has been evolved throughout the years, and I wanna see how it will be in the future.

We have the temperature of the month from 1880 to 2022, so we are in front of a Time Series.

A time series is data ordered by time. Each entry is preceded and followed by another and has a timestamp that determines the order of the data. It us to extract valuable insights from temporal data and consists in analyzing and making predictions based on time-based patterns.

[1]:
```
import pandas as pd
import numpy as np
import scipy
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
from sklearn.metrics import mean_squared_error
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.graphics.tsaplots import plot_predict
from statsmodels.tsa.stattools import adfuller
```

1.1 Exploratory data analysis

[2]:
```
df_temperature = pd.read_csv("monthly_temperature.csv")
df_temperature.head(10)
```

[2]:

	Year	Month	Temperature
0	1880	1	-0.36
1	1881	1	-0.31
2	1882	1	0.26
3	1883	1	-0.58
4	1884	1	-0.17
5	1885	1	-1.01

```
 6  1886      1        -0.75
 7  1887      1        -1.08
 8  1888      1        -0.48
 9  1889      1        -0.28
```

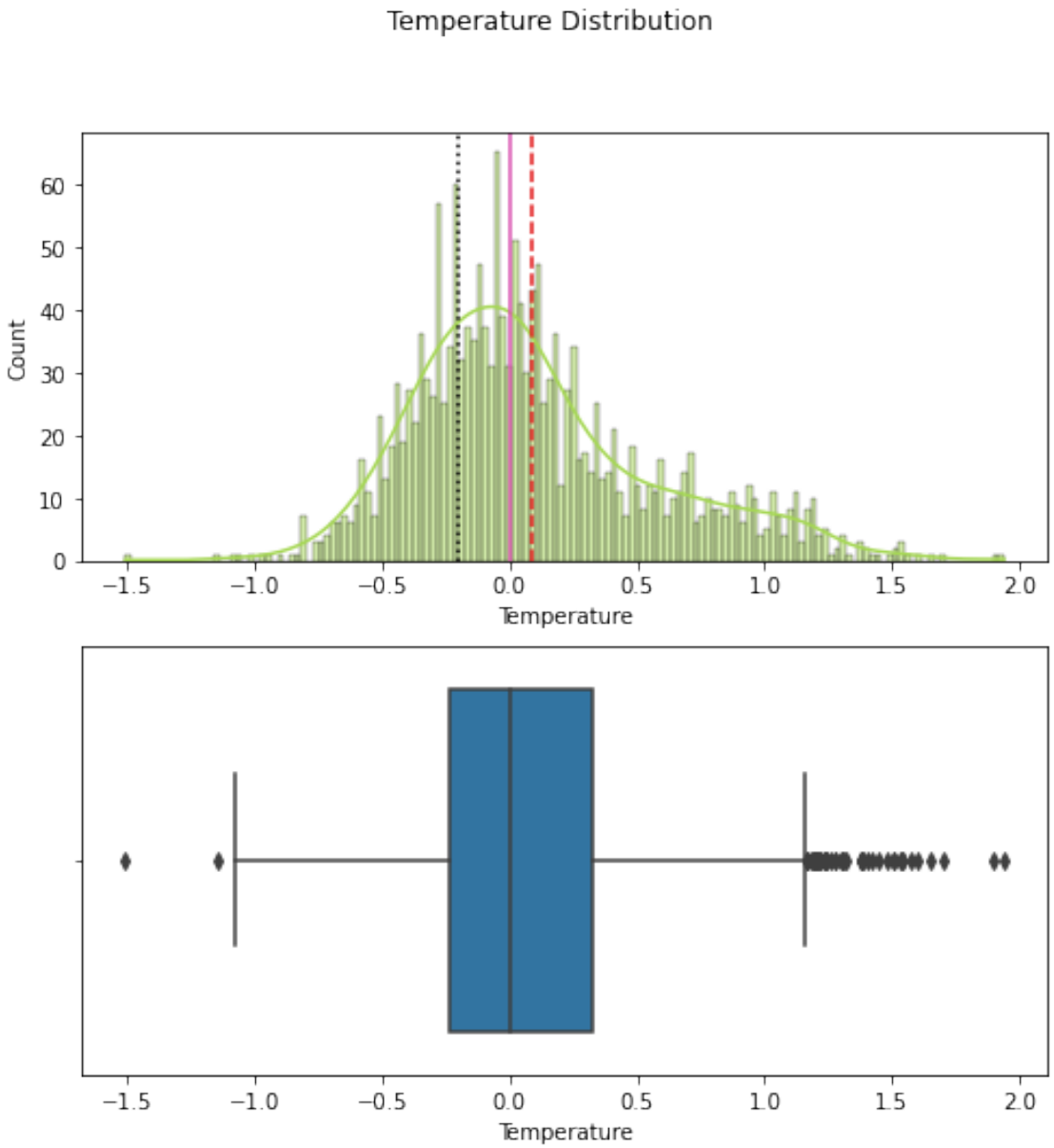
[3]:
```
df_temperature.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1716 entries, 0 to 1715
Data columns (total 3 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Year         1716 non-null   int64
 1   Month        1716 non-null   int64
 2   Temperature  1716 non-null   float64
dtypes: float64(1), int64(2)
memory usage: 40.3 KB
```

We have the temperature of 1716 months, and we don't have empty entries.

Let's see the behavior of the temperatures.

[4]:
```
fix, ax = plt.subplots(2, 1, figsize = (8, 8))
fix.suptitle("Temperature Distribution")
sns.histplot(data=df_temperature, x="Temperature", color="#a6d854", kde=True,␣
 ↪ax = ax[0], bins=150)
ax[0].axvline(x = df_temperature["Temperature"].mean(), color="#e31a1c",␣
 ↪ls="--")
ax[0].axvline(x = df_temperature["Temperature"].mode()[0], color="black", ls=":
 ↪")
ax[0].axvline(x = df_temperature["Temperature"].median(), color="#db57b2",␣
 ↪ls="-")
sns.boxplot(x=df_temperature["Temperature"], ax = ax[1])
_ = ax[1].set_xlabel("Temperature")
```



In black we can see the mode, in pink the median and the with red the mean.

[5]:
```
scipy.stats.skew(df_temperature["Temperature"])
```

[5]:
```
0.7702387649516262
```

[6]:
```
df_temperature["Temperature"].mode()
```

[6]:
```
0   -0.2
Name: Temperature, dtype: float64
```

[7]:
```
df_temperature["Temperature"].describe()
```

[7]:
```
count    1716.000000
mean        0.086603
std         0.475834
min        -1.510000
25%        -0.240000
50%         0.000000
75%         0.320000
max         1.940000
Name: Temperature, dtype: float64
```

We have 1716 months, 143 years from 1880 to 2022.

We see a left skew distribution, with many outliers with high values, indicating that these values should not be normal.

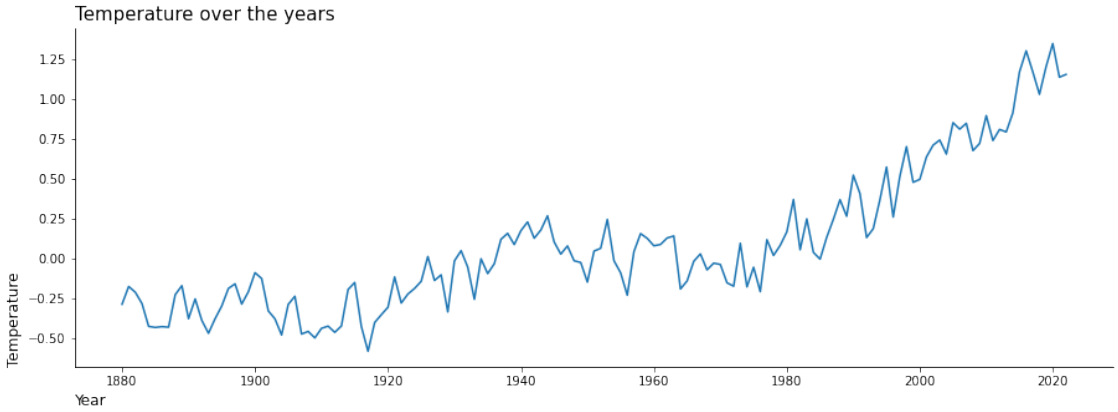
The median is 0. The minimum temperature is -1.51, the maximum is 1.94 and the mean 0.086. The most popular temperature is -0.2.

Let's explore now how the temperature has evolved over the years.

[8]:
```
mean_temp_year = df_temperature.groupby(['Year']).mean().reset_index()
mean_temp_year
```

[8]:
```
      Year  Month  Temperature
0     1880    6.5    -0.289167
1     1881    6.5    -0.177500
2     1882    6.5    -0.214167
3     1883    6.5    -0.285000
4     1884    6.5    -0.428333
..     …    …           …
138   2018    6.5     1.030833
139   2019    6.5     1.205000
140   2020    6.5     1.350000
141   2021    6.5     1.138333
142   2022    6.5     1.155833

[143 rows x 3 columns]
```

[9]:
```
fig, ax = plt.subplots(figsize=(15, 5))
temp_years_line_plot = sns.lineplot(data=mean_temp_year, y="Temperature",␣
↪x="Year", ax=ax)
temp_years_line_plot.set_title('Temperature over the years', fontsize = 15,␣
↪loc="left")
temp_years_line_plot.set_xlabel('Year', fontsize=12, loc="left")
_ = temp_years_line_plot.set_ylabel('Temperature', fontsize=12, loc="bottom")
sns.despine()
```



We can see an upward trend of the temperature, and since 1990, the trend is getting stronger. We do not see many temperatures below zero since 1990.

[10]:
```
df_temperature[df_temperature["Temperature"] < 0].sort_values(by="Year",␣
↪ascending=False).head()
```

[10]:

	Year	Month	Temperature
257	1994	2	-0.05
1543	1993	11	-0.21
1257	1993	9	-0.07
970	1992	7	-0.11
1542	1992	11	-0.09

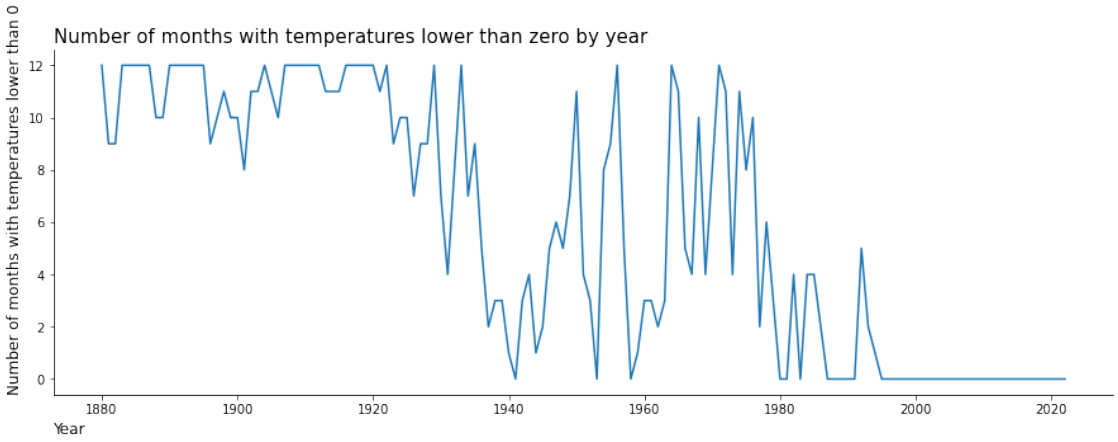
The last temperature below zero was on February 1994.

Let's see now how many times the temperature was below zero each year, and its behavior.

[11]:
```
df_below_zero_temp = df_temperature[df_temperature["Temperature"] < 0].
↪groupby("Year").count().reset_index()
df_below_zero_temp = df_below_zero_temp.rename(columns={"Temperature":␣
↪"Count_Below_0"})

#I do a merge to get the months without below zero temperatures
df_count_b0 = pd.merge(mean_temp_year, df_below_zero_temp, on="Year",␣
↪how="outer")
df_count_b0["Count_Below_0"] = df_count_b0["Count_Below_0"].fillna(value=0)
```

[12]:
```
fig, ax = plt.subplots(figsize=(15, 5))
temp_below_zero_line_plot = sns.lineplot(data=df_count_b0, y="Count_Below_0",␣
↪x="Year", ax=ax)
temp_below_zero_line_plot.set_title('Number of months with temperatures lower␣
↪than zero by year', fontsize = 15, loc="left")
temp_below_zero_line_plot.set_xlabel('Year', fontsize=12, loc="left")
```

```
_ = temp_below_zero_line_plot.set_ylabel('Number of months with temperatures␣
↪lower than 0', fontsize=12, loc="bottom")
sns.despine()
```



Here we can see the number of below zero temperatures over the years. The count varies with the years.

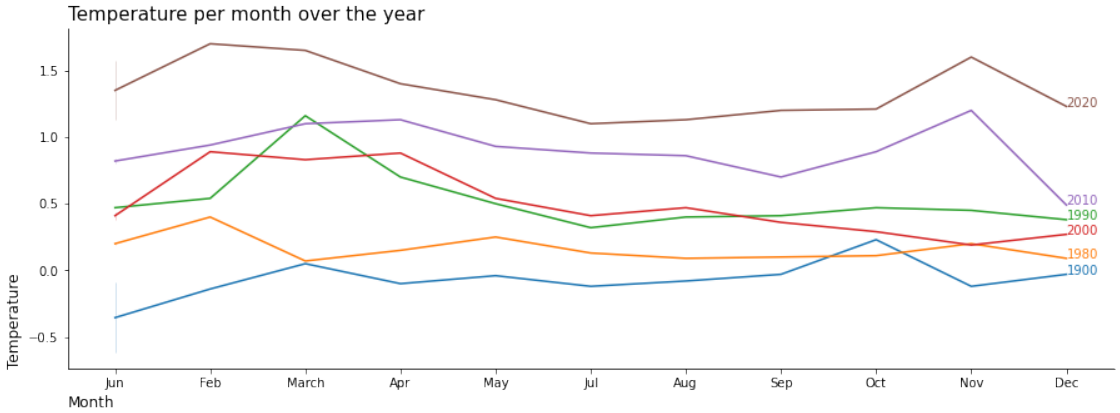
We observe that there were few months with below zero temperatures around of 1940 and 1960. And the last year with below zero temperature was in 1994 where there was just one month.

Now, let's see how the temperature has varied in each month through the years.

[13]:
```
def month_description(col):
    if col["Month"] == 1:
        return "Jun"
    elif col["Month"] == 2:
        return "Feb"
    elif col["Month"] == 3:
        return "March"
    elif col["Month"] == 4:
        return "Apr"
    elif col["Month"] == 5:
        return "May"
    elif col["Month"] == 6:
        return "Jun"
    elif col["Month"] == 7:
        return "Jul"
    elif col["Month"] == 8:
        return "Aug"
    elif col["Month"] == 9:
        return "Sep"
    elif col["Month"] == 10:
        return "Oct"
```

```
    elif col["Month"] == 11:
        return "Nov"
    return "Dec"

df_temperature["MonthDesc"] = df_temperature.apply(month_description, axis=1)
```

[14]:
```
fig, ax = plt.subplots(figsize=(15, 5))
df_temp_filter = df_temperature[ (df_temperature["Year"] == 1900) |␣
↪(df_temperature["Year"] == 1980) |
                                 (df_temperature["Year"] == 1990) |␣
↪(df_temperature["Year"] == 2000) |
                                 (df_temperature["Year"] == 2010) |␣
↪(df_temperature["Year"] == 2020) ]

years = df_temp_filter['Year'].unique()
palette = sns.color_palette("tab10", n_colors=len(years))
temp_months_years_plot = sns.lineplot(data=df_temp_filter,
                                      y="Temperature", x="MonthDesc",␣
↪hue="Year", ax=ax, palette=palette, legend=None)
for year in years:
    year_data = df_temp_filter[df_temp_filter['Year'] == year]
    line_color = palette[years.tolist().index(year)]  # Get the corresponding␣
↪line color
    plt.text(year_data['MonthDesc'].iloc[-1], year_data['Temperature'].
↪iloc[-1], year, color=line_color)

temp_months_years_plot.set_title('Temperature per month over the year',␣
↪fontsize = 15, loc="left")
temp_months_years_plot.set_xlabel('Month', fontsize=12, loc="left")
_ = temp_months_years_plot.set_ylabel('Temperature', fontsize=12, loc="bottom")
sns.despine()
```



In the lines we see some peaks in March and other in November. We can appreciate also the seasonal pattern.

Also we can observe that the temperature has increased with strength after 1980. Looking the the lines of the years 1900 and 1980, with 80 years of difference, we see that are close to each other. On the other hand, if we see the lines of 2010 and 2022, these are more separated and just in 10 years.

1.2 Modeling

The data set represents a time series, as we have the month and the temperature of the month.

I will analyze it by applying models for time series and try to predict future values.

First at all, I will set the index.

[15]:
```
# Combine "Year" and "Month" columns into a single datetime column
data = df_temperature.copy()
data["Date"] = pd.to_datetime(data["Year"].astype(str) + "-" + data["Month"].
 ↪astype(str))
data = data.sort_values("Date")
# Set the "Date" column as the index of the DataFrame
data.set_index("Date", inplace=True)

# Drop the original "Year" and "Month" columns
data.drop(columns=["Year", "Month", "MonthDesc"], inplace=True)
data
```

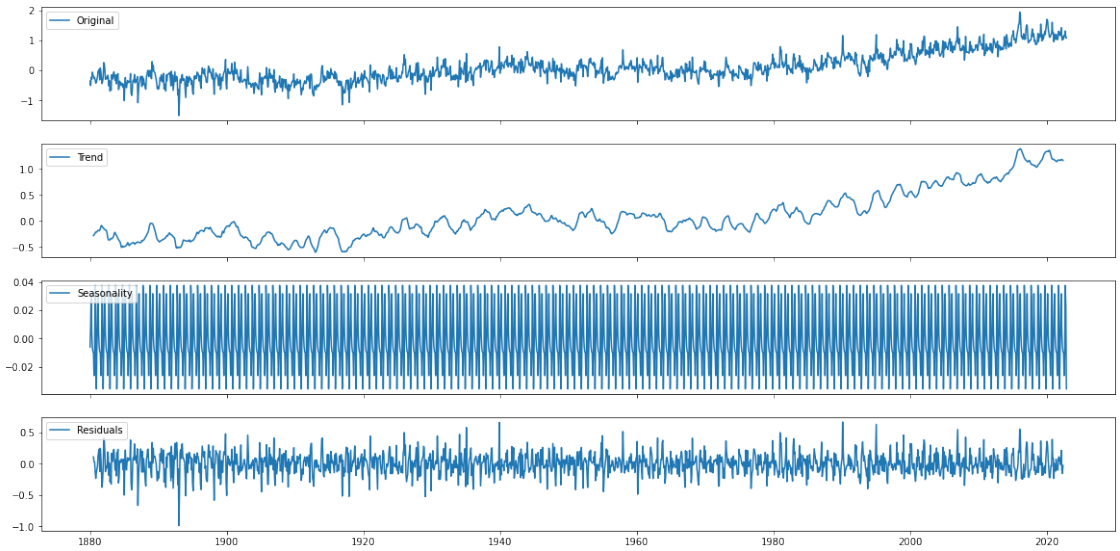
[15]:
```
            Temperature
Date
1880-01-01        -0.36
1880-02-01        -0.51
1880-03-01        -0.23
1880-04-01        -0.30
1880-05-01        -0.06
…                 …
2022-08-01         1.16
2022-09-01         1.15
2022-10-01         1.31
2022-11-01         1.08
2022-12-01         1.08

[1716 rows x 1 columns]
```

Let's split the serie in their components, a time serie has four components: * Trend: A long-term movement of the time series, such as the decreasing average heart rate of workouts as a person gets fitter. * Seasonality: Regular periodic occurrences within a time interval smaller than a year. * Irregularity: Short-term irregular fluctuations or noise. * Cyclicity: Repeated fluctuations around

the trend that are longer in duration than irregularities but shorter than what would constitute a trend

[16]:
```
plt.figure(figsize=(15, 5))
ts_dicomposition = seasonal_decompose(data, period=12)

trend_estimate = ts_dicomposition.trend
seasonal_estimate = ts_dicomposition.seasonal
residual_estimate = ts_dicomposition.resid

fig, axes = plt.subplots(4, 1, sharex=True, sharey=False)
fig.set_figheight(10)
fig.set_figwidth(20)
# First plot to the Original time series
axes[0].plot(data, label='Original')
axes[0].legend(loc='upper left');
# second plot to be for trend
axes[1].plot(trend_estimate, label='Trend')
axes[1].legend(loc='upper left');
# third plot to be Seasonality component
axes[2].plot(seasonal_estimate, label='Seasonality')
axes[2].legend(loc='upper left');
# last last plot to be Residual component
axes[3].plot(residual_estimate, label='Residuals')
axes[3].legend(loc='upper left');
```

```
<Figure size 1080x360 with 0 Axes>
```

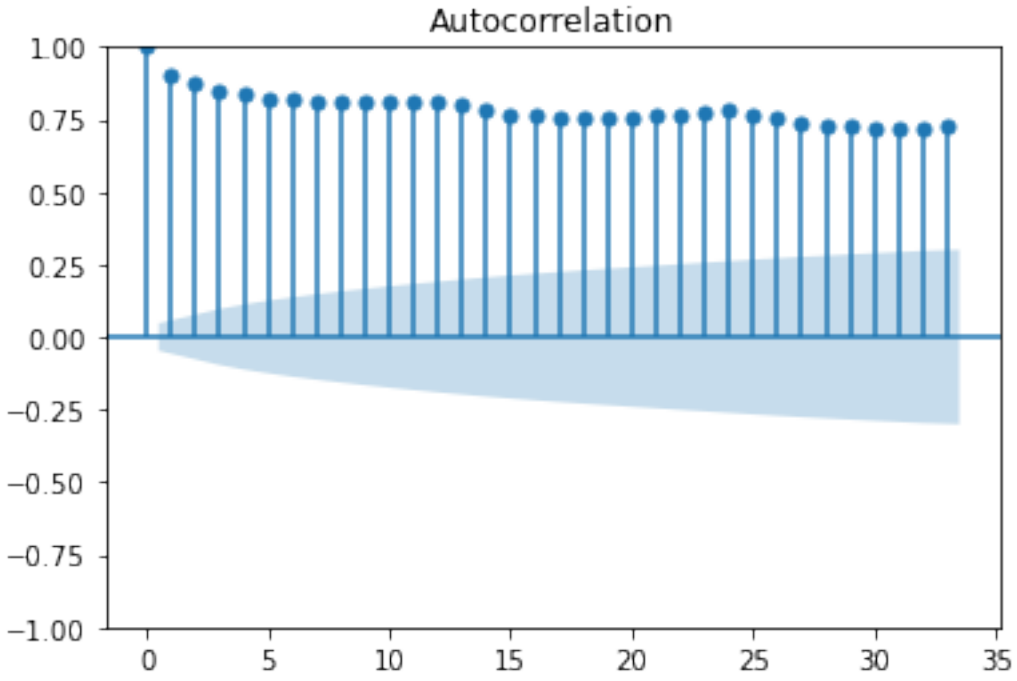


We can appreciate the an upward trend, and the repated seasonality.

As we have negative temperature values, we can say that it is an additive model.

No let's check the autorrelation in order to check the seasonality.

[17]:
```
plot_acf(data["Temperature"]);
```



In this ACF we can see how the peaks are every 12 months, showing the seasonality.

Before starting the modeling, we have to check the stationarity. This is a way to measure if the data has structural patterns like seasonal trends.

Stationarity means that the manner in which time series data changes is constant. A stationary time series will not have any trends or seasonal patterns. The statistical properties of the series like mean, variance and autocorrelation are constant over time.

Stationarity is assumed for a wide variety of time series forecasting methods including autoregressive moving average like SARIMA.

To check this, we can use ADF test.

[18]:
```
# Extract the temperature data as a one-dimensional array (Series)
temperature_series = data["Temperature"]

# Perform the ADF test
result = adfuller(temperature_series)

adf_statistic = result[0]
p_value = result[1]
```

```
critical_values = result[4]

# Results
print("ADF Statistic:", adf_statistic)
print("P-value:", p_value)
```

```
ADF Statistic: -0.5124219970407082
P-value: 0.8895695466735319
```

As the p-value is higher than 0.05 I cannot reject the null hypothesis, so the data is non-stationary.

To make it stationary we can: - Differencing the Series (once or more) - Take the log of the series (Transformation) - Detrending

In this case, I will apply differencing in order to achieve stationarity. I start with 1 month.

[19]:
```
temperature_series = data["Temperature"]
temperature_1_month_diff = temperature_series.diff(1).dropna()

# Perform the ADF test
result = adfuller(temperature_1_month_diff)

p_value = result[1]

# Results
print("ADF Statistic:", adf_statistic)
print("P-value:", p_value)
```

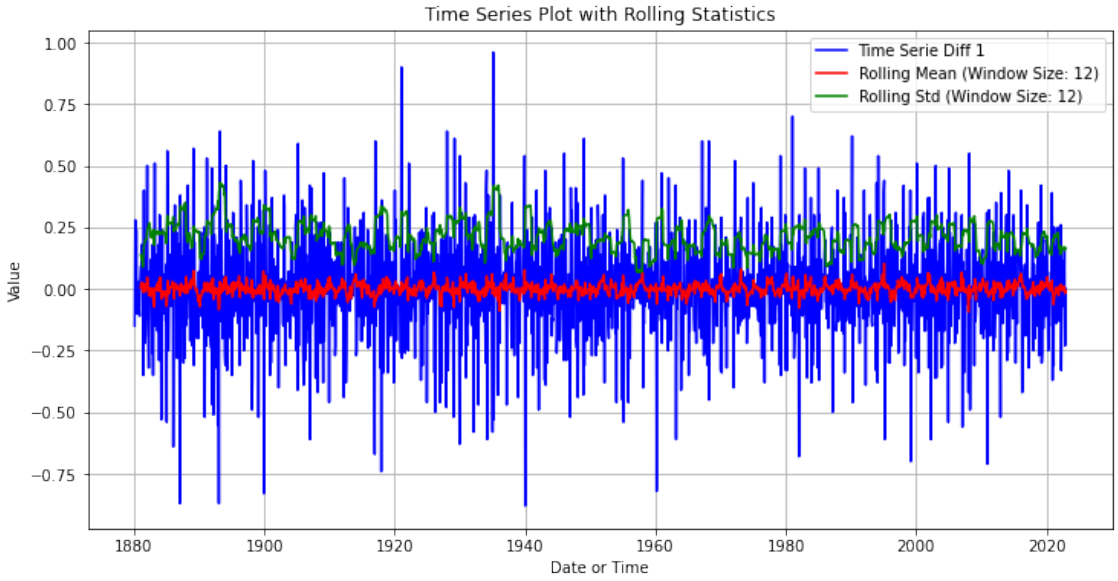
```
ADF Statistic: -0.5124219970407082
P-value: 2.301875987630783e-25
```

Now that p-value is lower than 0.05, we can reject the null hypothesis and we get the stationarity.

I will check that with a plot.

[20]:
```
window_size = 12
rolling_mean = temperature_1_month_diff.rolling(window=window_size).mean()
rolling_std = temperature_1_month_diff.rolling(window=window_size).std()

plt.figure(figsize=(12, 6))
plt.plot(temperature_1_month_diff, label='Time Serie Diff 1', color='blue')
plt.plot(rolling_mean, label=f'Rolling Mean (Window Size: {window_size})',␣
↪color='red')
plt.plot(rolling_std, label=f'Rolling Std (Window Size: {window_size})',␣
↪color='green')
plt.xlabel("Date or Time")
plt.ylabel("Value")
plt.title("Time Series Plot with Rolling Statistics")
plt.legend()
plt.grid(True)
```

```
plt.show()
```



Here we can confirm that the mean and standard deviation are constans.

1.2.1 Models parameters

As the serie has a seasonal pattern, I will use SARIMA(p, d, q)x(P, D, Q, m) model.

The SARIMA model (Seasonal ARIMA) is an extension of the ARIMA model. This is achieved by adding a linear combination of seasonal past values and forecast errors.

In order to use the model, we need to find the values of the parameters p, d, q and P, D, Q. The last three are related to the seasonal component.

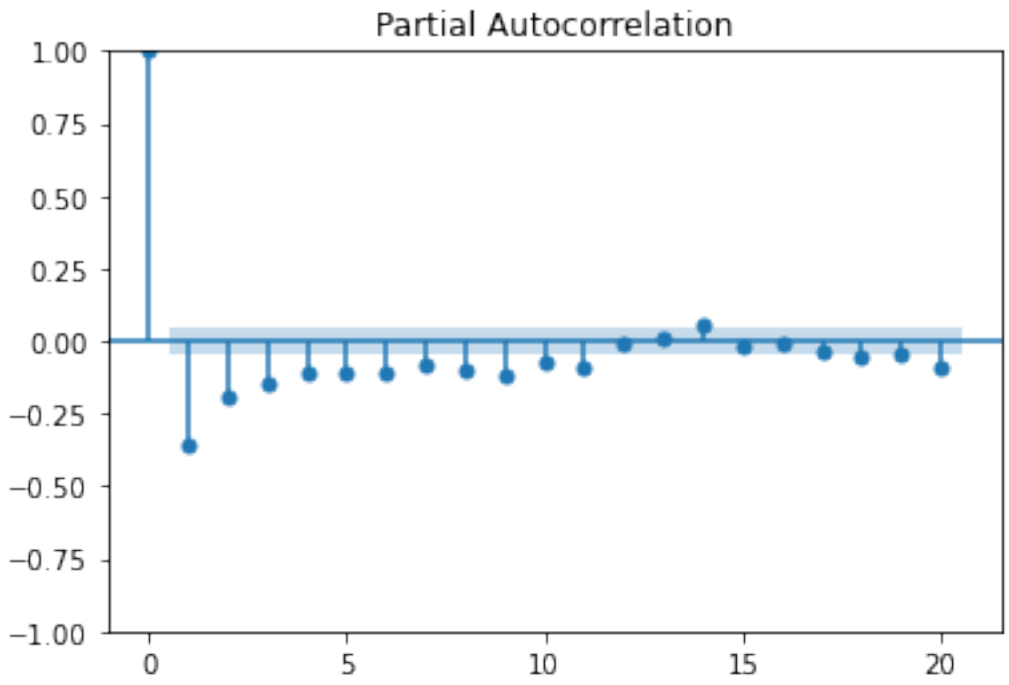
Parameter d The parameter **d** represents the number of times data is differenced to make it stationary, I we have achieved the stationarity by differencing 1.

Parameter p It is the number of lag observations, It refers to the number of lags to be used as predictors.

For the **p** value, I will check the Partial Autocorrelation (PACF) plot.

[21]:
```
plot_pacf(data["Temperature"].diff(1).dropna(), lags=20);
```

```
C:\Users\ouw-Alejandro.Sandle\Anaconda3\lib\site-
packages\statsmodels\graphics\tsaplots.py:348: FutureWarning: The default method
'yw' can produce PACF values outside of the [-1,1] interval. After 0.13, the
default will change tounadjusted Yule-Walker ('ywm'). You can use this method
now by setting method='ywm'.
  warnings.warn(
```



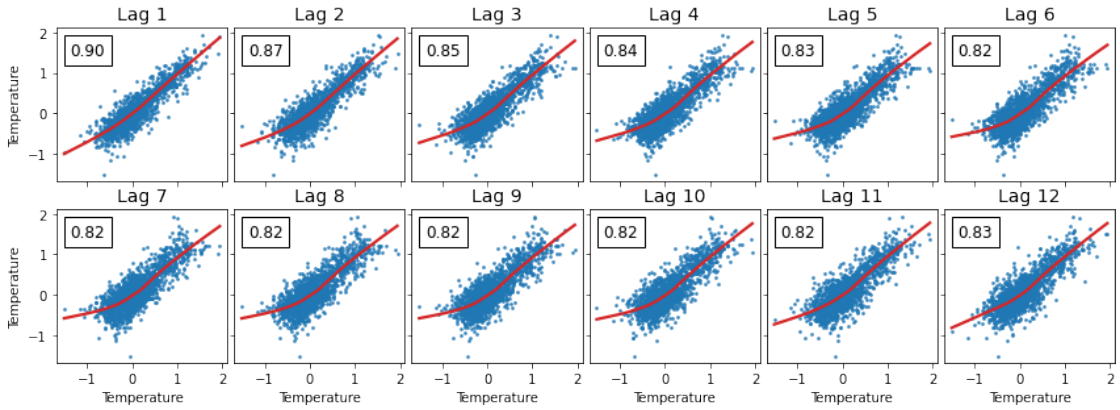
We can see that the peak that extends beyond the significance region is with lag 1. The magnitude or height of the peak indicates the strength of the partial autocorrelation. Larger peaks represent stronger autocorrelations, and they are more likely to be relevant for estimating the order of the autoregressive component.

So I will select 1 for **p**.

We can confirm that with a lag plot.

[22]:
```
def lagplot(x, y=None, lag=1, standardize=False, ax=None, **kwargs):
    from matplotlib.offsetbox import AnchoredText
    x_ = x.shift(lag)
    if standardize:
        x_ = (x_ - x_.mean()) / x_.std()
    if y is not None:
        y_ = (y - y.mean()) / y.std() if standardize else y
    else:
        y_ = x
    corr = y_.corr(x_)
    if ax is None:
        fig, ax = plt.subplots()
    scatter_kws = dict(
        alpha=0.75,
        s=3,
    )
    line_kws = dict(color='C3', )
```

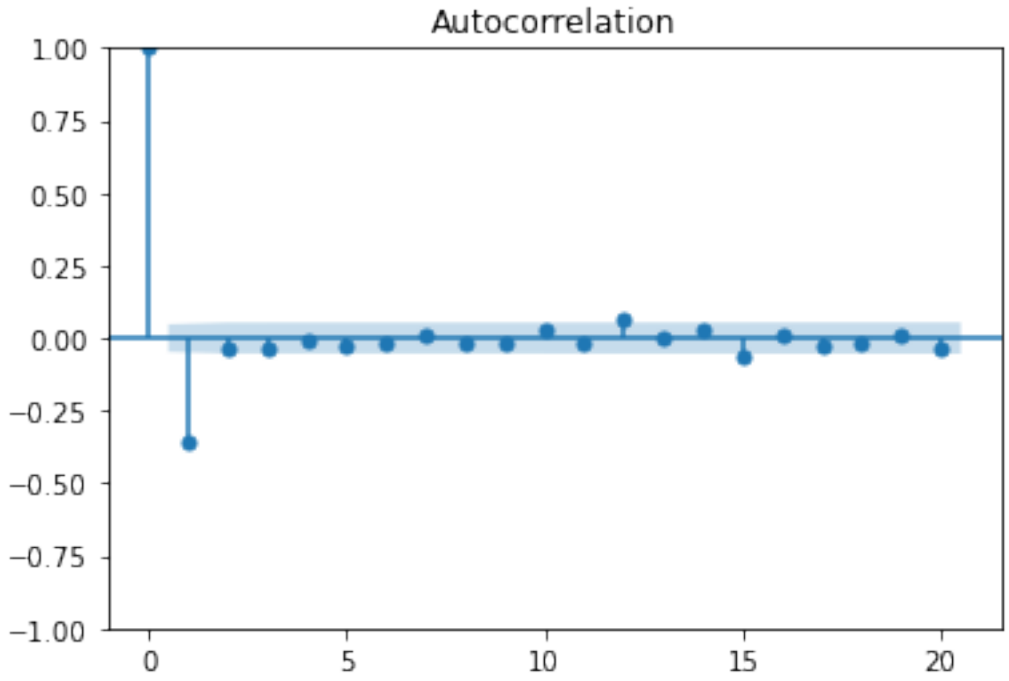
```
    ax = sns.regplot(x=x_,
                     y=y_,
                     scatter_kws=scatter_kws,
                     line_kws=line_kws,
                     lowess=True,
                     ax=ax,
                     **kwargs)
    at = AnchoredText(
        f"{corr:.2f}",
        prop=dict(size="large"),
        frameon=True,
        loc="upper left",
    )
    at.patch.set_boxstyle("square, pad=0.0")
    ax.add_artist(at)
    ax.set(title=f"Lag {lag}", xlabel=x_.name, ylabel=y_.name)
    return ax

def plot_lags(x, y=None, lags=6, nrows=1, lagplot_kwargs={}, **kwargs):
    import math
    kwargs.setdefault('nrows', nrows)
    kwargs.setdefault('ncols', math.ceil(lags / nrows))
    kwargs.setdefault('figsize', (kwargs['ncols'] * 2, nrows * 2 + 0.5))
    fig, axs = plt.subplots(sharex=True, sharey=True, squeeze=False, **kwargs)
    for ax, k in zip(fig.get_axes(), range(kwargs['nrows'] * kwargs['ncols'])):
        if k + 1 <= lags:
            ax = lagplot(x, y, lag=k + 1, ax=ax, **lagplot_kwargs)
            ax.set_title(f"Lag {k + 1}", fontdict=dict(fontsize=14))
            ax.set(xlabel="", ylabel="")
        else:
            ax.axis('off')
    plt.setp(axs[-1, :], xlabel=x.name)
    plt.setp(axs[:, 0], ylabel=y.name if y is not None else x.name)
    fig.tight_layout(w_pad=0.1, h_pad=0.1)
    return fig

plot_lags(data.Temperature, lags=12, nrows=2);
```



We can see that we have more correlation with lag 1.

Parameter q To find de **q** value, I will take a look to the ACF plot.

[23]:
```
plot_acf(data["Temperature"].diff().dropna().values, lags=20);
```



Here clearly we can see the peak in lag 1.

So, We have the values of the three parameters (1, 1, 1).

Now let's see the seasonal parameters.

1.2.2 Parameter m

We have seen that the series has a season of 12 months.

In order to find the other parameters we can apply the similar methods than before.

1.2.3 Parameter D

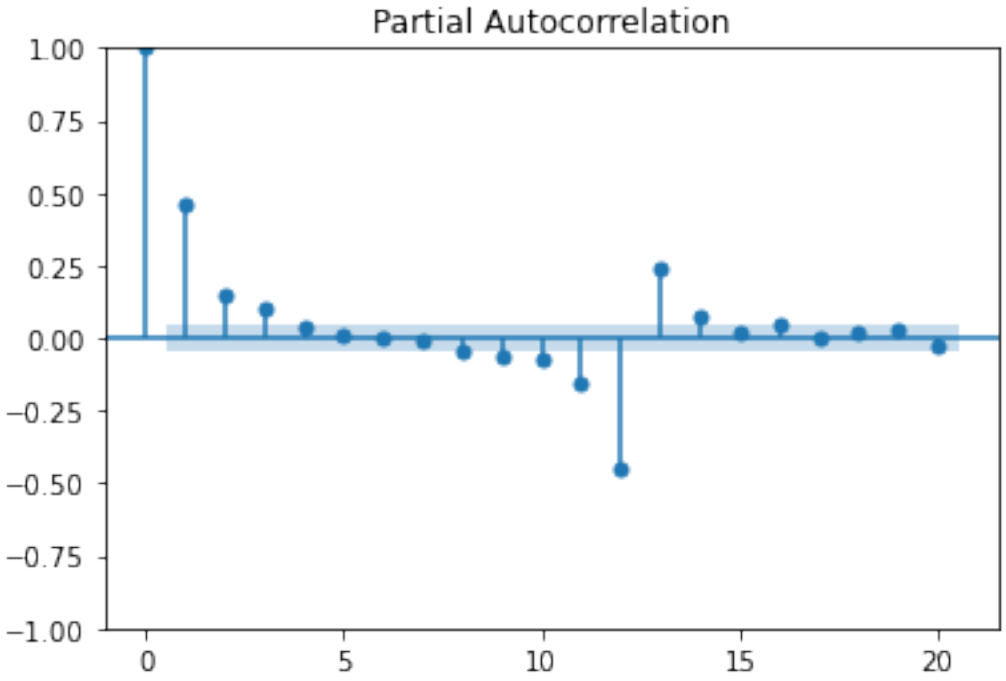
As a general rule, set the model parameters such that D never exceeds one. And the total differencing 'd + D' never exceeds 2, and as our model has well defined seasonal patterns, I will set it with 1.

1.2.4 Parameter Q

We can use the PACF plot to find Q, by looking for significant spikes at seasonal lags, which are multiples of the seasonality period. These significant spikes indicate partial autocorrelations at the seasonal lags, representing the direct influence of past observations from the same season on the current value

[24]:
```
plot_pacf(data["Temperature"].diff(12).dropna().values, lags=20);
```

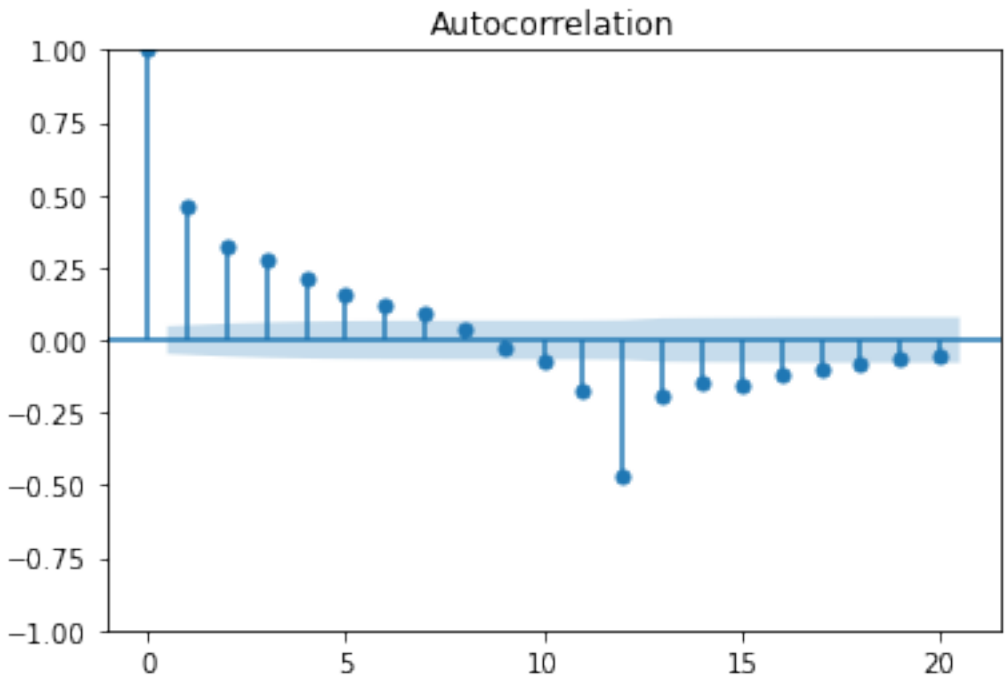
```
C:\Users\ouw-Alejandro.Sandle\Anaconda3\lib\site-
packages\statsmodels\graphics\tsaplots.py:348: FutureWarning: The default method
'yw' can produce PACF values outside of the [-1,1] interval. After 0.13, the
default will change tounadjusted Yule-Walker ('ywm'). You can use this method
now by setting method='ywm'.
  warnings.warn(
```



We see a peak in lag 1.

1.2.5 Parameter P

[25]:
```
plot_acf(data["Temperature"].diff(12).dropna().values, lags=20);
```



Here we see also peak in lag 1.

Now that we have all the parameters, we have to take care that all these values are estimations, we should test and evaluate the model with other values also.

[26]:
```
model = sm.tsa.statespace.SARIMAX(data['Temperature'], order=(1, 1, 1),␣
 ↪seasonal_order=(0, 1, 1, 12))
results = model.fit()
print(results.summary())
```

```
C:\Users\ouw-Alejandro.Sandle\Anaconda3\lib\site-
packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency MS will be used.
  self._init_dates(dates, freq)
C:\Users\ouw-Alejandro.Sandle\Anaconda3\lib\site-
packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency MS will be used.
  self._init_dates(dates, freq)

                                 SARIMAX Results
==========================================================================
```

```
==========
Dep. Variable:                        Temperature   No. Observations:
1716
Model:             SARIMAX(1, 1, 1)x(0, 1, 1, 12)   Log Likelihood
465.647
Date:                            Fri, 20 Oct 2023   AIC
-923.294
Time:                                    15:13:13   BIC
-901.534
Sample:                                01-01-1880   HQIC
-915.239
                                     - 12-01-2022
Covariance Type:                              opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.3052      0.028     10.923      0.000       0.250       0.360
ma.L1         -0.8562      0.017    -50.977      0.000      -0.889      -0.823
ma.S.L12      -0.9413      0.008   -121.964      0.000      -0.956      -0.926
sigma2         0.0333      0.001     37.927      0.000       0.032       0.035
===================================================================================
===
Ljung-Box (L1) (Q):                   0.85   Jarque-Bera (JB):
146.22
Prob(Q):                              0.36   Prob(JB):
0.00
Heteroskedasticity (H):               0.78   Skew:
0.07
Prob(H) (two-sided):                  0.00   Kurtosis:
4.43
===================================================================================
===

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

[27]:
```
#lets take the last 40 years as test set.
train = data.iloc[:len(data) - 480] #480
test = data.iloc[len(data) - 480:]

model = sm.tsa.statespace.SARIMAX(train, order=(1, 1, 1), seasonal_order=(1, 1,␣
 ↪1, 12))

model_fit = model.fit()
predictions = model_fit.forecast(len(test))
print(mean_squared_error(test["Temperature"], predictions, squared=False))
```

```
C:\Users\ouw-Alejandro.Sandle\Anaconda3\lib\site-
packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency MS will be used.
  self._init_dates(dates, freq)
C:\Users\ouw-Alejandro.Sandle\Anaconda3\lib\site-
packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency MS will be used.
  self._init_dates(dates, freq)
```

```
0.5038470976926622
```

We see a p-value higher than 0.05 in ar.S.L12, let's changes this value and test it again.

[28]:
```
model = sm.tsa.statespace.SARIMAX(data['Temperature'], order=(1, 1, 1),␣
↪seasonal_order=(0, 1, 1, 12))
results = model.fit()
print(results.summary())
```

```
C:\Users\ouw-Alejandro.Sandle\Anaconda3\lib\site-
packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency MS will be used.
  self._init_dates(dates, freq)
C:\Users\ouw-Alejandro.Sandle\Anaconda3\lib\site-
packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency MS will be used.
  self._init_dates(dates, freq)
```

```
                                     SARIMAX Results
==========================================================================================
==========
Dep. Variable:                         Temperature   No. Observations:
1716
Model:             SARIMAX(1, 1, 1)x(0, 1, 1, 12)   Log Likelihood
465.647
Date:                            Fri, 20 Oct 2023   AIC
-923.294
Time:                                    15:13:20   BIC
-901.534
Sample:                                01-01-1880   HQIC
-915.239
                                     - 12-01-2022
Covariance Type:                               opg
==========================================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------------------
ar.L1          0.3052      0.028     10.923      0.000       0.250       0.360
ma.L1         -0.8562      0.017    -50.977      0.000      -0.889      -0.823
ma.S.L12      -0.9413      0.008   -121.964      0.000      -0.956      -0.926
sigma2         0.0333      0.001     37.927      0.000       0.032       0.035
```

```
==========================================================================
===
Ljung-Box (L1) (Q):                   0.85   Jarque-Bera (JB):
146.22
Prob(Q):                              0.36   Prob(JB):
0.00
Heteroskedasticity (H):               0.78   Skew:
0.07
Prob(H) (two-sided):                  0.00   Kurtosis:
4.43
==========================================================================
===

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

[29]:
```
#lets take the last 40 years as test set.
train = data.iloc[:len(data) - 480] #480
test = data.iloc[len(data) - 480:]

model = sm.tsa.statespace.SARIMAX(train, order=(1, 1, 1), seasonal_order=(0, 1,␣
 ↪1, 12))

model_fit = model.fit()
predictions = model_fit.forecast(len(test))
print(mean_squared_error(test["Temperature"], predictions, squared=False))
```

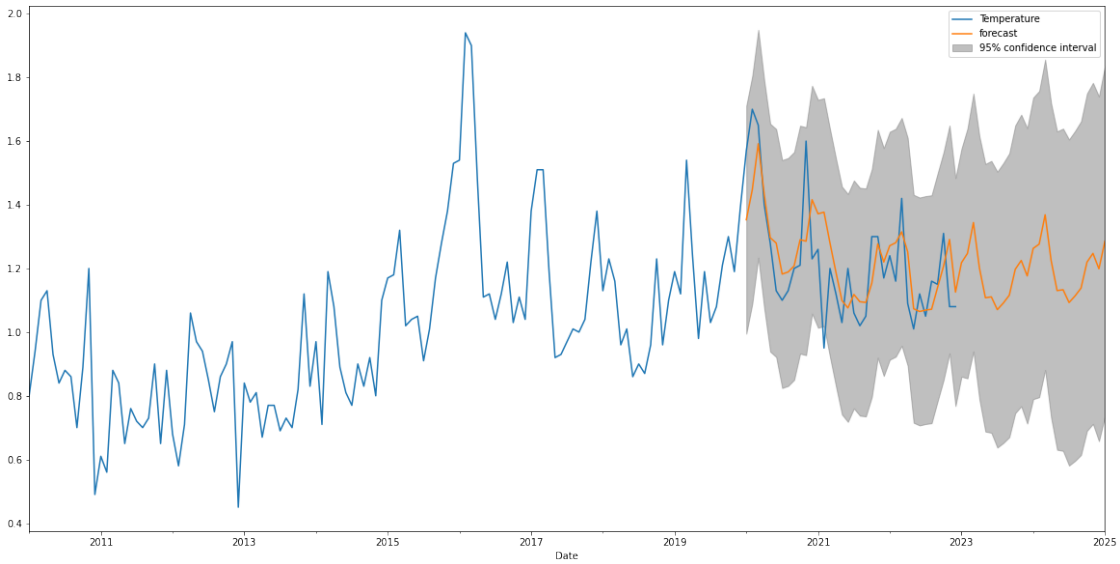
```
C:\Users\ouw-Alejandro.Sandle\Anaconda3\lib\site-
packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency MS will be used.
  self._init_dates(dates, freq)
C:\Users\ouw-Alejandro.Sandle\Anaconda3\lib\site-
packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency MS will be used.
  self._init_dates(dates, freq)

0.4959548376732838
```

With this change we get all the parameters with p-value less than 0.05, and lower RMSE and AIC.

As we have p-value of ar.S.L12 higher than 0.05, let's see what happen if we remove the P value.

Now let's compare the actual serie with the stimated, and some future values.

[30]:
```
fig, ax = plt.subplots(figsize=(20, 10))
ax = data.loc['2010':].plot(ax=ax)
plot_predict(results, start='2020', end='2025', ax=ax)
plt.show()
```

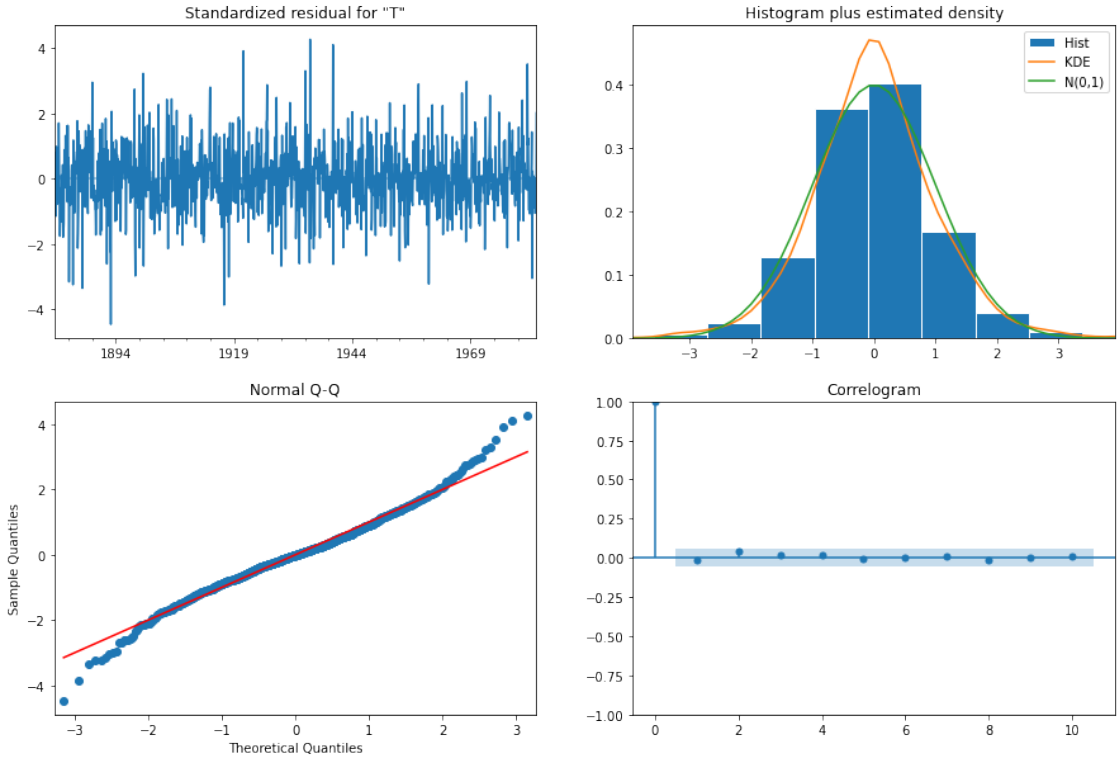


The predicted series is similar to the actual, and the future trend continue in the upward way.

Let's get now the MSE error, for that, I will take the las 40 years as test set.

To end up, let's take a look to the residuals.

[31]:
```
model_fit.plot_diagnostics(figsize=(15,10));
```



- The residual errors seem to fluctuate around a mean of zero and have a uniform variance.
- The residual plot suggest a normal distribution with mean zero
- Most of the dots fall in the red line
- The residual erros are not correlated, and alla are within the range.
- And we get a RMSE of 0.49.

1.3 Temperature Prediction

As the last temperature is 2022-12-01, what would be the temperature in the next month, 2023-01-01 with a confident of 95%?

[32]:
```
model_trained = sm.tsa.statespace.SARIMAX(data, order=(1, 1, 1),␣
 ↪seasonal_order=(0, 1, 1, 12))
model_trained_fit = model_trained.fit()

preds_df = (model_trained_fit
            .get_prediction(start='2023-01-01', end='2023-01-01')
            .summary_frame(alpha=0.05)
)
print(preds_df.head())
```

```
C:\Users\ouw-Alejandro.Sandle\Anaconda3\lib\site-
packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency MS will be used.
  self._init_dates(dates, freq)
C:\Users\ouw-Alejandro.Sandle\Anaconda3\lib\site-
packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency MS will be used.
  self._init_dates(dates, freq)
```

Temperature	mean	mean_se	mean_ci_lower	mean_ci_upper
2023-01-01	1.217989	0.182615	0.86007	1.575907

We can see that the model predicted a temperature of 1.21 for 01-01-2023.