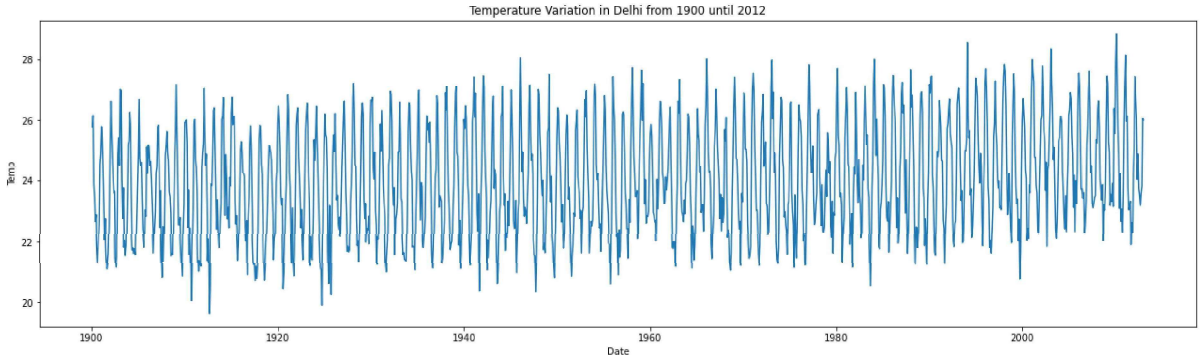
In [1]:
```
# First import the essential libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from sklearn.metrics import mean_squared_error
from math import sqrt
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

In [9]:
```
# Reading and transforming the file
cities = pd.read_csv('input/GlobalLandTemperaturesByCity.csv')
delhi = cities.loc[cities['City'] == 'Delhi', ['dt','AverageTemperature']]
delhi.columns = ['Date','Temp']
delhi['Date'] = pd.to_datetime(delhi['Date'])
delhi.reset_index(drop=True, inplace=True)
delhi.set_index('Date', inplace=True)

#I'm going to consider the temperature just from 1900 until the end of 2012
delhi = delhi.loc['1900':'2013-01-01']
delhi = delhi.asfreq('M', method='bfill')
delhi.head()
```

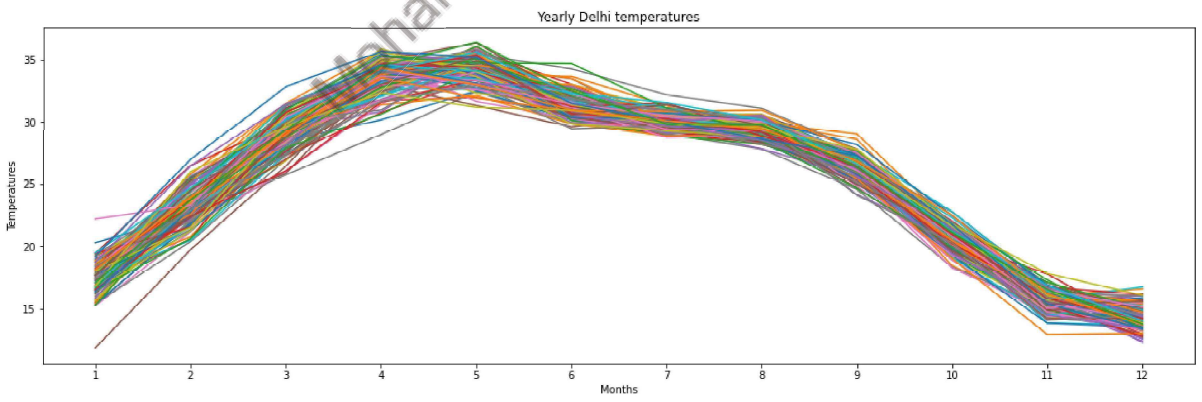
Out[9]:

Date	Temp
1900-01-31	17.648
1900-02-28	25.467
1900-03-31	28.327
1900-04-30	32.911
1900-05-31	35.305

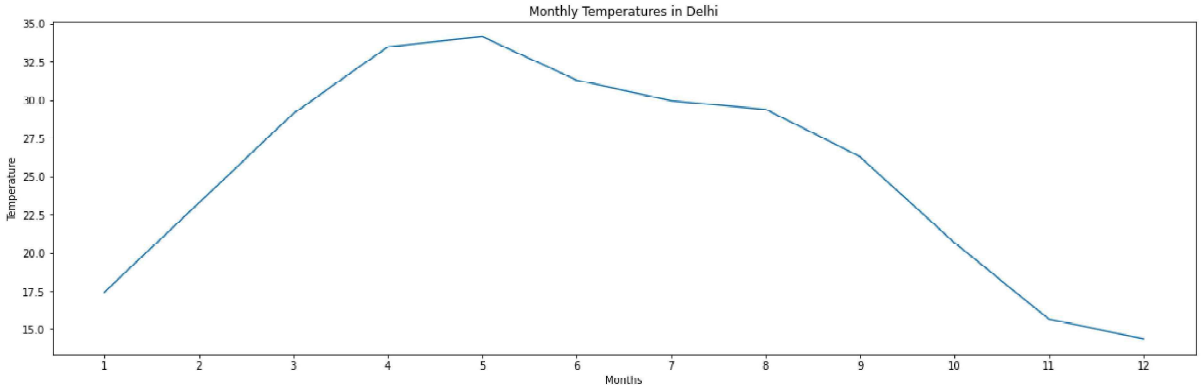
In [10]:
```
# Plot the series and check how it behaves
plt.figure(figsize=(22,6))
sns.lineplot(x=delhi.index, y=rio['Temp'])
plt.title('Temperature Variation in Delhi from 1900 until 2012')
plt.show()
```



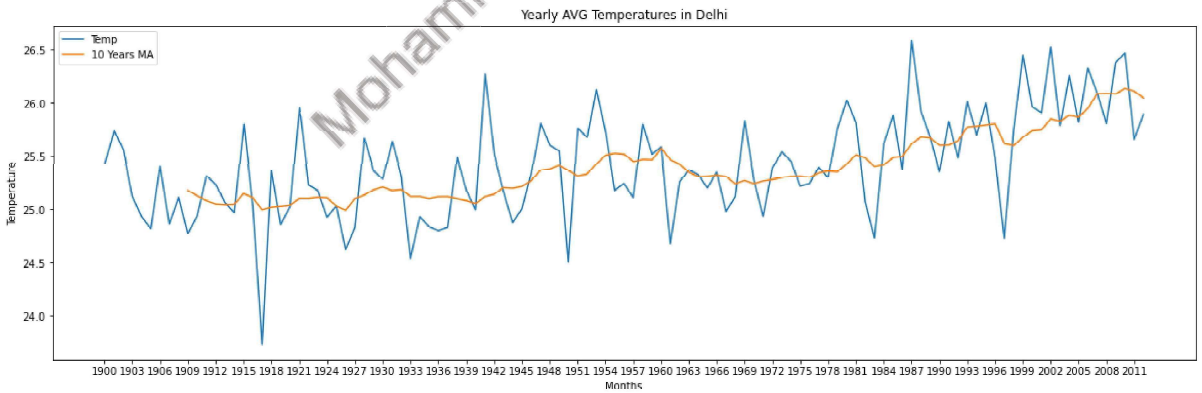
In [11]:
```
# Create a pivot table to plot the monthly temperatures through the years
delhi['month'] = delhi.index.month
delhi['year'] = delhi.index.year
pivot = pd.pivot_table(delhi, values='Temp', index='month', columns='year', ag
gfunc='mean')
pivot.plot(figsize=(20,6))
plt.title('Yearly Delhi temperatures')
plt.xlabel('Months')
plt.ylabel('Temperatures')
plt.xticks([x for x in range(1,13)])
plt.legend().remove()
plt.show()
```

In [12]:
```
monthly_seasonality = pivot.mean(axis=1)
monthly_seasonality.plot(figsize=(20,6))
plt.title('Monthly Temperatures in Delhi')
plt.xlabel('Months')
plt.ylabel('Temperature')
plt.xticks([x for x in range(1,13)])
plt.show()
```



In [14]:
```
year_avg = pd.pivot_table(delhi, values='Temp', index='year', aggfunc='mean')
year_avg['10 Years MA'] = year_avg['Temp'].rolling(10).mean()
year_avg[['Temp','10 Years MA']].plot(figsize=(20,6))
plt.title('Yearly AVG Temperatures in Delhi')
plt.xlabel('Months')
plt.ylabel('Temperature')
plt.xticks([x for x in range(1900,2012,3)])
plt.show()
```

In [15]:
```
# Split the data in training, validation and test set
train = delhi[:-60].copy()
val = delhi[-60:-12].copy()
test = delhi[-12:].copy()
```

In [16]:
```
# Excluding the first line, as it has NaN values
baseline = val['Temp'].shift()
baseline.dropna(inplace=True)
baseline.head()
```

Out[16]:
```
Date
2008-02-29    16.809
2008-03-31    25.979
2008-04-30    29.609
2008-05-31    32.313
2008-06-30    31.151
Freq: M, Name: Temp, dtype: float64
```

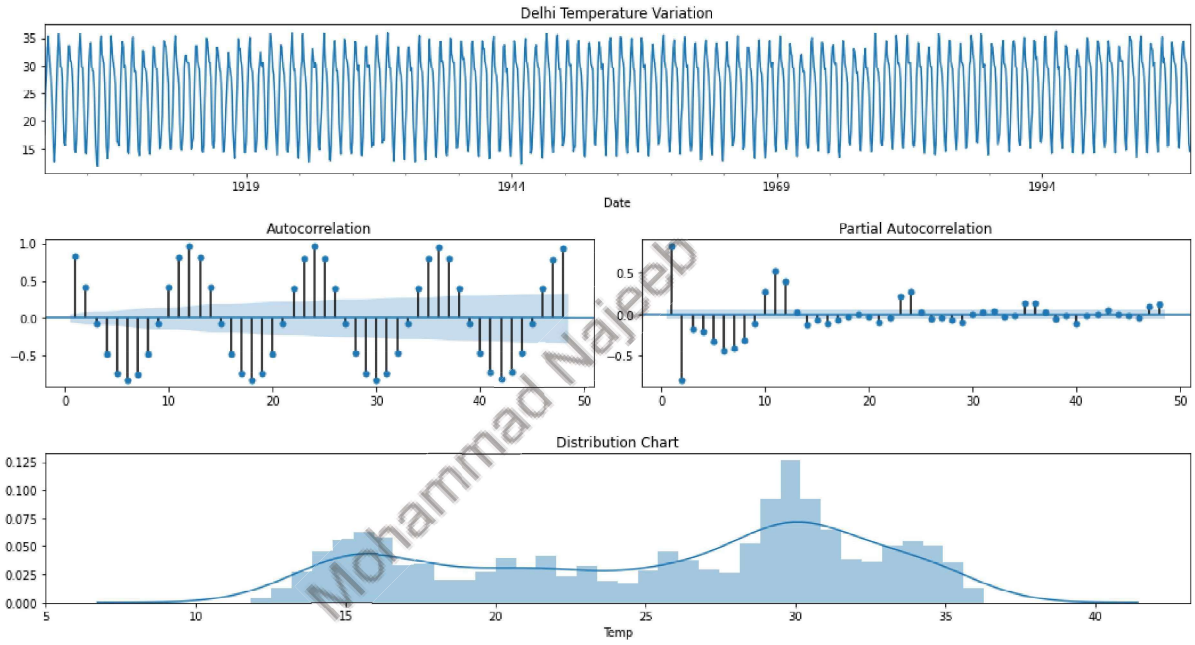
In [17]:
```
def measure_rmse(y_true, y_pred):
    return sqrt(mean_squared_error(y_true,y_pred))

# Using the function with the baseline values
rmse_base = measure_rmse(val.iloc[1:,0],baseline)
print(f'The RMSE of the baseline that we will try to diminish is {round(rmse_b
ase,4)} celsius degrees')
```

```
The RMSE of the baseline that we will try to diminish is 3.981 celsius degree
s
```

In [18]:
```
def check_stationarity(y, lags_plots=48, figsize=(22,8)):
    "Use Series as parameter"

    # Creating plots of the DF
    y = pd.Series(y)
    fig = plt.figure()

    ax1 = plt.subplot2grid((3, 3), (0, 0), colspan=2)
    ax2 = plt.subplot2grid((3, 3), (1, 0))
    ax3 = plt.subplot2grid((3, 3), (1, 1))
    ax4 = plt.subplot2grid((3, 3), (2, 0), colspan=2)

    y.plot(ax=ax1, figsize=figsize)
    ax1.set_title('Delhi Temperature Variation')
    plot_acf(y, lags=lags_plots, zero=False, ax=ax2);
    plot_pacf(y, lags=lags_plots, zero=False, ax=ax3);
    sns.distplot(y, bins=int(sqrt(len(y))), ax=ax4)
    ax4.set_title('Distribution Chart')

    plt.tight_layout()

    print('Results of Dickey-Fuller Test:')
    adfinput = adfuller(y)
    adftest = pd.Series(adfinput[0:4], index=['Test Statistic','p-value','Lags
Used','Number of Observations Used'])
    adftest = round(adftest,4)

    for key, value in adfinput[4].items():
        adftest["Critical Value (%s)"%key] = value.round(4)

    print(adftest)

    if adftest[0].round(2) < adftest[5].round(2):
        print('\nThe Test Statistics is lower than the Critical Value of 5%.\n
The serie seems to be stationary')
    else:
        print("\nThe Test Statistics is higher than the Critical Value of 5%.
\nThe serie isn't stationary")
```

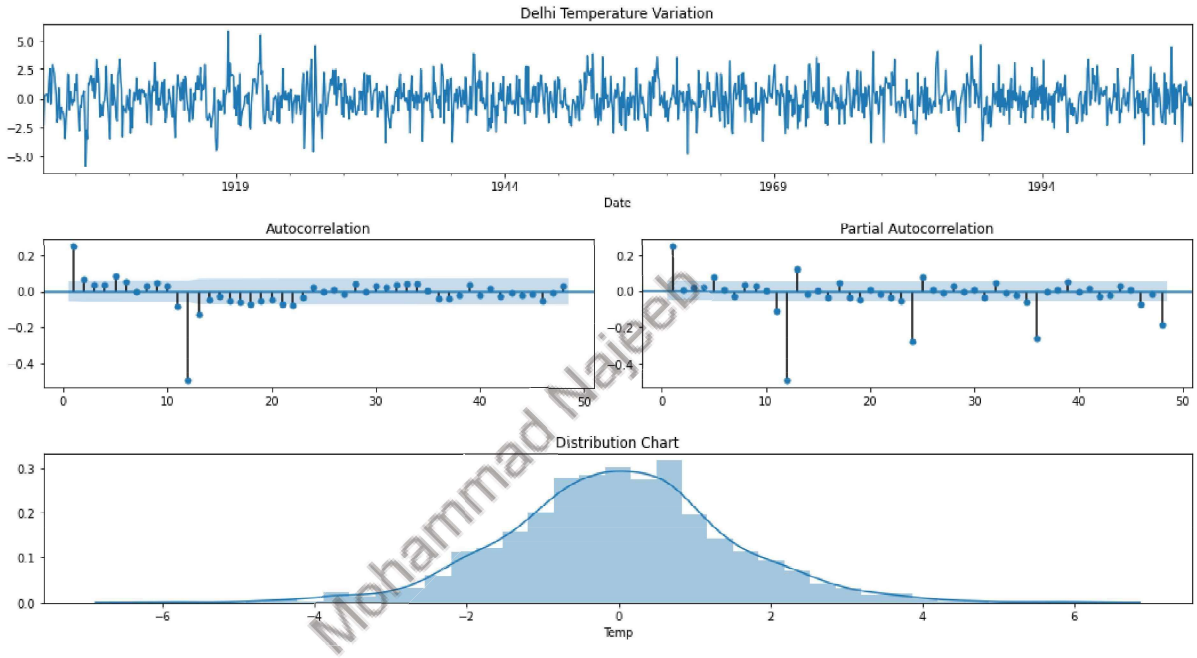
In [19]:
```
# The first approach is to check the series without any transformation
check_stationarity(train['Temp'])
```

```
Results of Dickey-Fuller Test:
Test Statistic                  -4.6402
p-value                          0.0001
Lags Used                       23.0000
Number of Observations Used    1272.0000
Critical Value (1%)              -3.4355
Critical Value (5%)              -2.8638
Critical Value (10%)             -2.5680
dtype: float64

The Test Statistics is lower than the Critical Value of 5%.
The serie seems to be stationary
```



In [20]:
```
check_stationarity(train['Temp'].diff(12).dropna())
```

```
Results of Dickey-Fuller Test:
Test Statistic                  -12.4752
p-value                           0.0000
Lags Used                        23.0000
Number of Observations Used    1260.0000
Critical Value (1%)              -3.4356
Critical Value (5%)              -2.8638
Critical Value (10%)             -2.5680
dtype: float64

The Test Statistics is lower than the Critical Value of 5%.
The serie seems to be stationary
```



In [22]:
```
def walk_forward(training_set, validation_set, params):
    '''
    Params: it's a tuple where you put together the following SARIMA parameter
s: ((pdq), (PDQS), trend)
    '''
    history = [x for x in training_set.values]
    prediction = list()

    # Using the SARIMA parameters and fitting the data
    pdq, PDQS, trend = params

    #Forecasting one period ahead in the validation set
    for week in range(len(validation_set)):
        model = sm.tsa.statespace.SARIMAX(history, order=pdq, seasonal_order=P
DQS, trend=trend)
        result = model.fit(disp=False)
        yhat = result.predict(start=len(history), end=len(history))
        prediction.append(yhat[0])
        history.append(validation_set[week])

    return prediction
```

In [23]:
```
# Let's test it in the validation set
val['Pred'] = walk_forward(train['Temp'], val['Temp'], ((3,0,0),(0,1,1,12),'c'
))
```

In [24]:
```
# Measuring the error of the prediction
rmse_pred = measure_rmse(val['Temp'], val['Pred'])

print(f"The RMSE of the SARIMA(3,0,0),(0,1,1,12),'c' model was {round(rmse_pre
d,4)} celsius degrees")
print(f"It's a decrease of {round((rmse_pred/rmse_base-1)*100,2)}% in the RMS
E")
```

```
The RMSE of the SARIMA(3,0,0),(0,1,1,12),'c' model was 1.0978 celsius degrees
It's a decrease of -72.42% in the RMSE
```

In [25]:
```
# Creating the error column
val['Error'] = val['Temp'] - val['Pred']
```

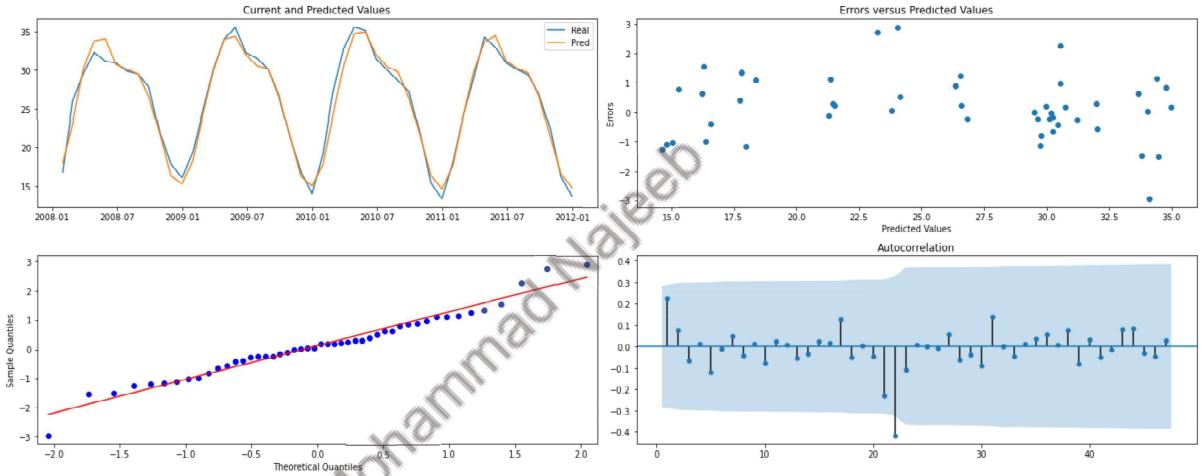
In [26]:
```
def plot_error(data, figsize=(20,8)):
    '''
    There must have 3 columns following this order: Temperature, Prediction, E
rror
    '''
    plt.figure(figsize=figsize)
    ax1 = plt.subplot2grid((2,2), (0,0))
    ax2 = plt.subplot2grid((2,2), (0,1))
    ax3 = plt.subplot2grid((2,2), (1,0))
    ax4 = plt.subplot2grid((2,2), (1,1))

    #Plotting the Current and Predicted values
    ax1.plot(data.iloc[:,0:2])
    ax1.legend(['Real', 'Pred'])
    ax1.set_title('Current and Predicted Values')

    # Residual vs Predicted values
    ax2.scatter(data.iloc[:,1], data.iloc[:,2])
    ax2.set_xlabel('Predicted Values')
    ax2.set_ylabel('Errors')
    ax2.set_title('Errors versus Predicted Values')

    ## QQ Plot of the residual
    sm.graphics.qqplot(data.iloc[:,2], line='r', ax=ax3)

    # Autocorrelation plot of the residual
    plot_acf(data.iloc[:,2], lags=(len(data.iloc[:,2])-1),zero=False, ax=ax4)
    plt.tight_layout()
    plt.show()
```

In [27]:
```
# We need to remove some columns to plot the charts
val.drop(['month','year'], axis=1, inplace=True)
val.head()
```

Out[27]:

Date	Temp	Pred	Error
2008-01-31	16.809	17.989900	-1.180900
2008-02-29	25.979	23.246814	2.732186
2008-03-31	29.609	30.263343	-0.654343
2008-04-30	32.313	33.801418	-1.488418
2008-05-31	31.151	34.103117	-2.952117

In [28]:
```
plot_error(val)
```



In [29]:
```
#Creating the new concatenating the training and validation set:
future = pd.concat([train['Temp'], val['Temp']])
future.head()
```

Out[29]:
```
Date
1900-01-31    17.648
1900-02-28    25.467
1900-03-31    28.327
1900-04-30    32.911
1900-05-31    35.305
Name: Temp, dtype: float64
```

In [30]:
```
# Using the same parameters of the fitted model
model = sm.tsa.statespace.SARIMAX(future, order=(3,0,0), seasonal_order=(0,1,1
,12), trend='c')
result = model.fit(disp=False)
```

In [31]:
```
test['Pred'] = result.predict(start=(len(future)), end=(len(future)+13))
```

In [32]:
```
test[['Temp', 'Pred']].plot(figsize=(22,6))
plt.title('Current Values compared to the Extrapolated Ones')
plt.show()
```