Importing the Necessary Libraries

```python
#importing libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px

import warnings
warnings.filterwarnings('ignore')
```

Reading the Dataset using Pandas

```python
# importing dataset
data = pd.read_csv('C://Users//user//Downloads//GlobalTemperatures.csv')
data.head()
```

]:

	dt	LandAverageTemperature	LandAverageTemperatureUncertainty	LandMaxTemperature	LandMaxTemperatureUncertainty	LandMinTemperature	LandM
0	1750-01-01	3.034	3.574	NaN	NaN	NaN	
1	1750-02-01	3.083	3.702	NaN	NaN	NaN	
2	1750-03-01	5.626	3.076	NaN	NaN	NaN	
3	1750-04-01	8.490	2.451	NaN	NaN	NaN	
4	1750-05-01	11.573	2.072	NaN	NaN	NaN	

```python
#datatypes of data
data.dtypes
```

3]:

```
dt                                           object
LandAverageTemperature                      float64
LandAverageTemperatureUncertainty           float64
LandMaxTemperature                          float64
LandMaxTemperatureUncertainty               float64
LandMinTemperature                          float64
LandMinTemperatureUncertainty               float64
LandAndOceanAverageTemperature              float64
LandAndOceanAverageTemperatureUncertainty   float64
dtype: object
```

Checking for No of rows and Columns

```python
data.shape
```

]: (3192, 9)

```python
#information about the dataset
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3192 entries, 0 to 3191
Data columns (total 9 columns):
 #   Column                                     Non-Null Count  Dtype  
---  ------                                     --------------  -----  
 0   dt                                         3192 non-null   object 
 1   LandAverageTemperature                     3180 non-null   float64
 2   LandAverageTemperatureUncertainty          3180 non-null   float64
 3   LandMaxTemperature                         1992 non-null   float64
 4   LandMaxTemperatureUncertainty              1992 non-null   float64
 5   LandMinTemperature                         1992 non-null   float64
 6   LandMinTemperatureUncertainty              1992 non-null   float64
 7   LandAndOceanAverageTemperature             1992 non-null   float64
 8   LandAndOceanAverageTemperatureUncertainty  1992 non-null   float64
dtypes: float64(8), object(1)
memory usage: 224.6+ KB
```

```python
#description about the dataset
data.describe()
```

:

	LandAverageTemperature	LandAverageTemperatureUncertainty	LandMaxTemperature	LandMaxTemperatureUncertainty	LandMinTemperature	LandMinT
count	3180.000000	3180.000000	1992.000000	1992.000000	1992.000000	
mean	8.374731	0.938468	14.350601	0.479782	2.743595	
std	4.381310	1.096440	4.309579	0.583203	4.155835	
min	-2.080000	0.034000	5.900000	0.044000	-5.407000	
25%	4.312000	0.186750	10.212000	0.142000	-1.334500	
50%	8.610500	0.392000	14.760000	0.252000	2.949500	
75%	12.548250	1.419250	18.451500	0.539000	6.778750	
max	19.021000	7.880000	21.320000	4.373000	9.715000	

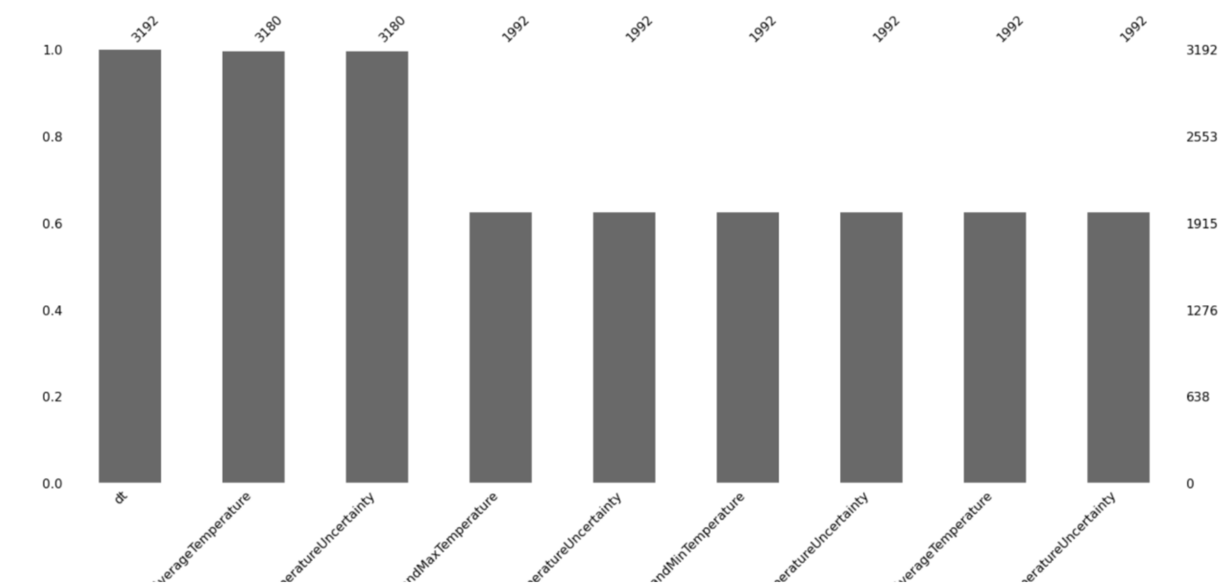
Checking the null values

```
data.isnull().sum()
```

```
dt                                        0
LandAverageTemperature                   12
LandAverageTemperatureUncertainty        12
LandMaxTemperature                     1200
LandMaxTemperatureUncertainty          1200
LandMinTemperature                     1200
LandMinTemperatureUncertainty          1200
LandAndOceanAverageTemperature         1200
LandAndOceanAverageTemperatureUncertainty    1200
dtype: int64
```

```
import missingno as msno
msno.bar(data)
```

```
<AxesSubplot:>
```



Exploratory data analysis

The exploratory data analysis is the approach of analyzing the complex datasets in order to summarize the main characteristics of the dataset. In general, the method of exploratory data analysis is used in different data visualization processes. Manu industries are included in the implementation of the exploratory data analysis for identifying the customer requirements.

Exploratory Data Analysis

Firstly,we seprate the year from the date column

10]:

```
data['dt'][0].split('-')[0]
```

Out[10]: '1750'

```
def fetch_year(date):
    return date.split('-')[0]
```

```
data['years']=data['dt'].apply(fetch_year)
```

Data represented in multiple columns

```
data.head()
```

	dt	LandAverageTemperature	LandAverageTemperatureUncertainty	LandMaxTemperature	LandMaxTemperatureUncertainty	LandMinTemperature	LandM
0	1750-01-01	3.034	3.574	NaN	NaN	NaN	
1	1750-02-01	3.083	3.702	NaN	NaN	NaN	
2	1750-03-01	5.626	3.076	NaN	NaN	NaN	
3	1750-04-01	8.490	2.451	NaN	NaN	NaN	
4	1750-05-01	11.573	2.072	NaN	NaN	NaN	

It is possible to get the meaningful insights by grouping the data year-wise

We group the data by years to get more meaningful insights.

```
data1=data.groupby('years').agg({'LandAverageTemperature':'mean','LandAverageTemperatureUncertainty':'mean'}).reset_inde
```

```
data1.head()
```

15]:

	years	LandAverageTemperature	LandAverageTemperatureUncertainty
0	1750	8.719364	2.637818
1	1751	7.976143	2.781143
2	1752	5.779833	2.977000
3	1753	8.388083	3.176000
4	1754	8.469333	3.494250

```python
data1['Uncertainity_top']= data1['LandAverageTemperature'] + data1['LandAverageTemperatureUncertainty']
data1['Uncertainity_bottom']= data1['LandAverageTemperature'] - data1['LandAverageTemperatureUncertainty']
```

```python
data1.head()
```

]:

	years	LandAverageTemperature	LandAverageTemperatureUncertainty	Uncertainity_top	Uncertainity_bottom
0	1750	8.719364	2.637818	11.357182	6.081545
1	1751	7.976143	2.781143	10.757286	5.195000
2	1752	5.779833	2.977000	8.756833	2.802833
3	1753	8.388083	3.176000	11.564083	5.212083
4	1754	8.469333	3.494250	11.963583	4.975083

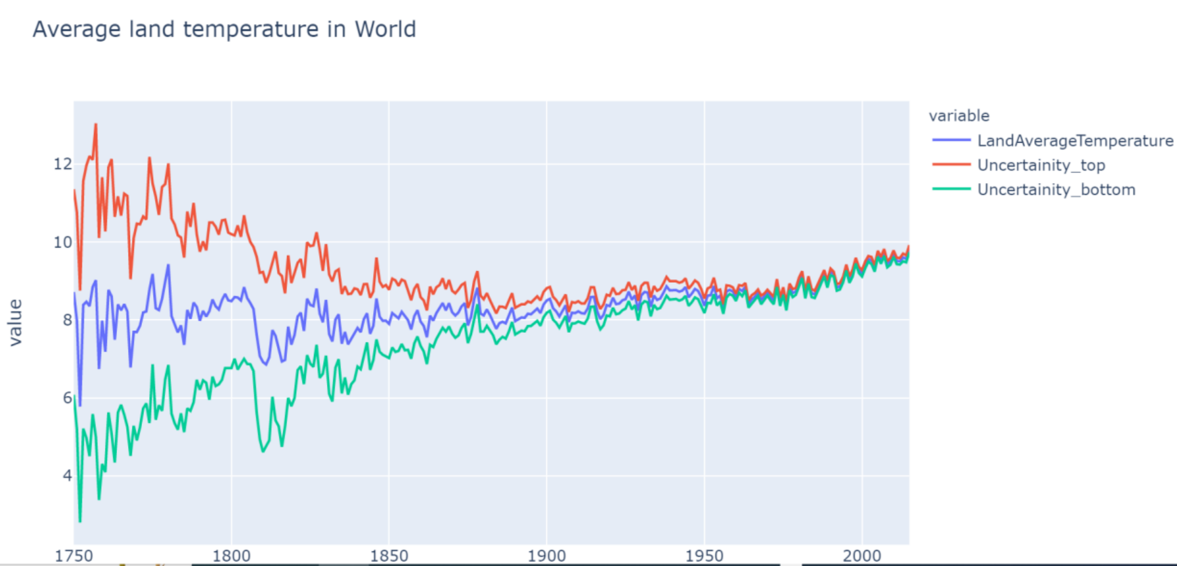
Names of all columns in the dataset

```python
data1.columns
```

:
```
Index(['years', 'LandAverageTemperature', 'LandAverageTemperatureUncertainty',
       'Uncertainity_top', 'Uncertainity_bottom'],
      dtype='object')
```

Graphical representation of the data

```python
fig = px.line(data1,x='years',y=['LandAverageTemperature','Uncertainity_top', 'Uncertainity_bottom'],title='Average land
fig.show()
```



From this we conclude,Global warming increases in the last decades,as average temperature is more .

Explore average temperature in each season

[20]:
```python
data['dt']=pd.to_datetime(data['dt'])
```

[21]:
```python
data.dtypes
```

Out[21]:
```
dt                                           datetime64[ns]
LandAverageTemperature                              float64
LandAverageTemperatureUncertainty                   float64
LandMaxTemperature                                  float64
LandMaxTemperatureUncertainty                       float64
LandMinTemperature                                  float64
LandMinTemperatureUncertainty                       float64
LandAndOceanAverageTemperature                      float64
LandAndOceanAverageTemperatureUncertainty           float64
years                                                object
dtype: object
```

```python
data['month']=data['dt'].dt.month
```

```python
data.head()
```

3]:

	dt	LandAverageTemperature	LandAverageTemperatureUncertainty	LandMaxTemperature	LandMaxTemperatureUncertainty	LandMinTemperature	LandM
0	1750-01-01	3.034	3.574	NaN	NaN	NaN	
1	1750-02-01	3.083	3.702	NaN	NaN	NaN	
2	1750-03-01	5.626	3.076	NaN	NaN	NaN	
3	1750-04-01	8.490	2.451	NaN	NaN	NaN	
4	1750-05-01	11.573	2.072	NaN	NaN	NaN	

finding the season

]:
```python
def get_season(month):
    if month>=3 and month<=5:
        return 'spring'
    elif month>=6 and month<=8:
        return 'summer'
    elif month>=9 and month<=11:
        return 'autumn'
    else:
        return 'winter'
```

]:
```python
data['season']=data['month'].apply(get_season)
```

```
data.head()
```

	dt	LandAverageTemperature	LandAverageTemperatureUncertainty	LandMaxTemperature	LandMaxTemperatureUncertainty	LandMinTemperature	LandM
0	1750-01-01	3.034	3.574	NaN	NaN	NaN	
1	1750-02-01	3.083	3.702	NaN	NaN	NaN	
2	1750-03-01	5.626	3.076	NaN	NaN	NaN	
3	1750-04-01	8.490	2.451	NaN	NaN	NaN	
4	1750-05-01	11.573	2.072	NaN	NaN	NaN	

```
years=data['years'].unique()
```

```
spring_temp = []
summer_temp = []
autumn_temp = []
winter_temp = []
for year in years:
    current_df=data[data['years'] == year]
    spring_temp.append(current_df[current_df['season'] == 'spring']['LandAverageTemperature'].mean())
    summer_temp.append(current_df[current_df['season'] == 'summer']['LandAverageTemperature'].mean())
    autumn_temp.append(current_df[current_df['season'] == 'autumn']['LandAverageTemperature'].mean())
    winter_temp.append(current_df[current_df['season'] == 'winter']['LandAverageTemperature'].mean())

```

```
season = pd.DataFrame()
```

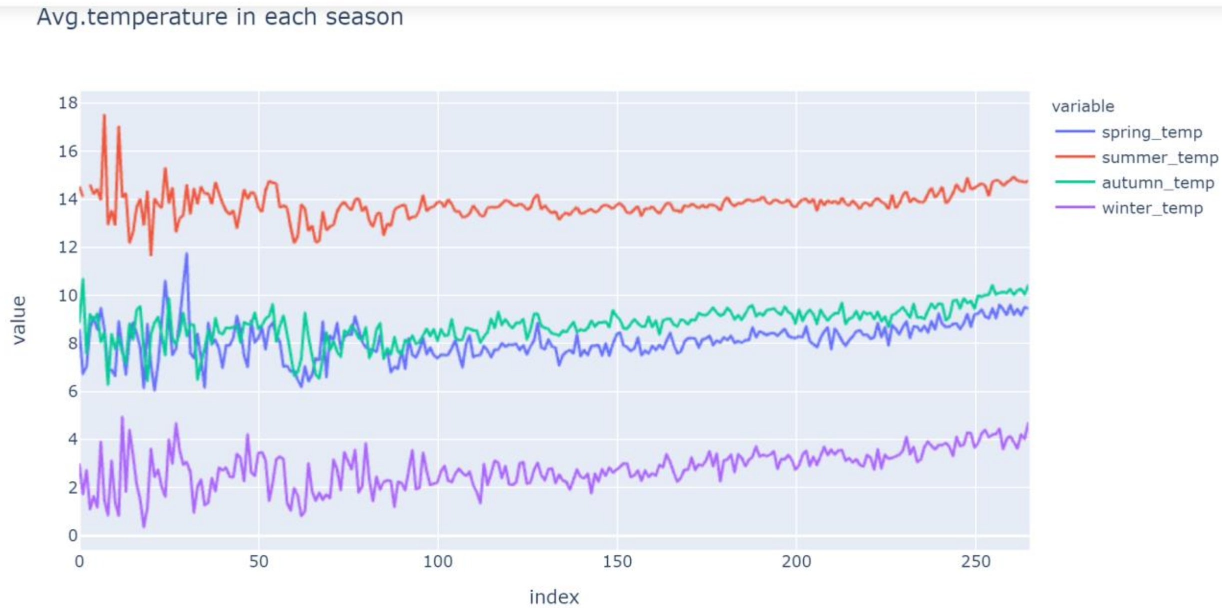
```
season['year'] = years
season['spring_temp'] = spring_temp
season['summer_temp'] = summer_temp
season['autumn_temp'] = autumn_temp
season['winter_temp'] = winter_temp
```

```
season.head()
```

	year	spring_temp	summer_temp	autumn_temp	winter_temp
0	1750	8.563000	14.518333	8.890000	2.963000
1	1751	6.735000	14.116000	10.673000	1.729000
2	1752	7.035500	NaN	7.587000	2.717000
3	1753	8.627333	14.608333	9.212333	1.104333
4	1754	9.074333	14.208333	8.957333	1.637333

```
fig=px.line(season,y=['spring_temp','summer_temp','autumn_temp','winter_temp'],title='Avg.temperature in each season')
fig.show()
```

Avg.temperature in each season



Data preprocessing

The data preprocessing is the method of preparing the raw data for the succeeding steps in the process of data analyzation. The data preprocessing is the preliminary step which is used in the process of data mining.

Season wise it's get warmer.

Data Preprocessing ¶

```
df=pd.read_csv('C://Users//user//Downloads//GlobalLandTemperaturesByCity.csv')
```

```
df.head()
```

34]:

	dt	AverageTemperature	AverageTemperatureUncertainty	City	Country	Latitude	Longitude
0	1743-11-01	6.068	1.737	Århus	Denmark	57.05N	10.33E
1	1743-12-01	NaN	NaN	Århus	Denmark	57.05N	10.33E
2	1744-01-01	NaN	NaN	Århus	Denmark	57.05N	10.33E
3	1744-02-01	NaN	NaN	Århus	Denmark	57.05N	10.33E
4	1744-03-01	NaN	NaN	Århus	Denmark	57.05N	10.33E

```
usa=df[df['Country']=='United States']
```

Checking on country USA

```
usa=df[df['Country']=='United States']
```

```
data=['New York','Los Angeles','San Francisco']
```

```
data2=usa[usa['City'].isin(data)]
data2.head()
```

]:

	dt	AverageTemperature	AverageTemperatureUncertainty	City	Country	Latitude	Longitude
4356748	1849-01-01	8.819	2.558	Los Angeles	United States	34.56N	118.70W
4356749	1849-02-01	9.577	1.970	Los Angeles	United States	34.56N	118.70W
4356750	1849-03-01	11.814	2.173	Los Angeles	United States	34.56N	118.70W
4356751	1849-04-01	13.704	2.902	Los Angeles	United States	34.56N	118.70W
4356752	1849-05-01	14.834	2.017	Los Angeles	United States	34.56N	118.70W

```
data2=data2[['dt','AverageTemperature']]
data2.head()
```

]:

	dt	AverageTemperature
4356748	1849-01-01	8.819
4356749	1849-02-01	9.577
4356750	1849-03-01	11.814
4356751	1849-04-01	13.704
4356752	1849-05-01	14.834

```
data2.columns=['Date','Temp']
data2.head()
```

	Date	Temp
4356748	1849-01-01	8.819
4356749	1849-02-01	9.577
4356750	1849-03-01	11.814
4356751	1849-04-01	13.704
4356752	1849-05-01	14.834

Checking on Null values

```
data2['Date']=pd.to_datetime(data2['Date'])
```

```
data2.isnull().sum()
```

```
]: Date       0
   Temp     120
   dtype: int64
```

Dropping Null Values and view the shape

```
data2.dropna(inplace=True)
```

```
data2.shape
```

```
]: (7073, 2)
```

Set Date as Index

```python
data2.set_index('Date',inplace=True)
```

Requirements for time series modelling


```python
data2.set_index('Date',inplace=True)
```


Whether it is stationary or not:

Conditions:

1.Time series should have a constant mean.

2.Time series should have a constant standard deviation.

3.Time series's auto-covariance should not depend on time.

Applying ADCF Test to check stationarity.

To check we use:

1.Rolling Statistics: Rolling statistics is a visualization technique, in which you plot the moving average to see if it varies over time.

2.ADCF Test: ADCF stands for Augmented Dickey-Fuller test which is a statistical unit root test. It gives us various values which can help us identifying stationarity. It comprises Test Statistics & some critical values for some confidence levels. If the Test statistics is less than the critical values, we can reject the null hypothesis & say that the series is stationary. The Null hypothesis says that time series is non-stationary. THE ADCF test also gives us a p-value. According to the null hypothesis, lower values of p is better.

```python
from statsmodels.tsa.stattools import adfuller
test_result=adfuller(data2['Temp'])
```

```python
def adfuller_test(temp):
    result=adfuller(temp)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations Used']
    for value,label in zip(result,labels):
        print(label+' : '+str(value) )
    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data has no unit root and is
    else:
        print("weak evidence against null hypothesis, time series has a unit root, indicating it is non-stationary ")
```

```
adfuller_test(data2['Temp'])
```

```
ADF Test Statistic : -2.0063893036758143
p-value : 0.28377865833331783
#Lags Used : 35
Number of Observations Used : 7037
weak evidence against null hypothesis, time series has a unit root, indicating it is non-stationary
```

From the result we have seen that it is non Stationary.

```
df=data2.copy()
```

```
df.head()
```

Date	Temp
1849-01-01	8.819
1849-02-01	9.577
1849-03-01	11.814
1849-04-01	13.704
1849-05-01	14.834

```
df['First_temp_diff']=df['Temp']-df['Temp'].shift(12)
```

```
df.head()
```

Date	Temp	First_temp_diff
1849-01-01	8.819	NaN
1849-02-01	9.577	NaN
1849-03-01	11.814	NaN
1849-04-01	13.704	NaN
1849-05-01	14.834	NaN

After Dropping NAN we have seen that it is stationary

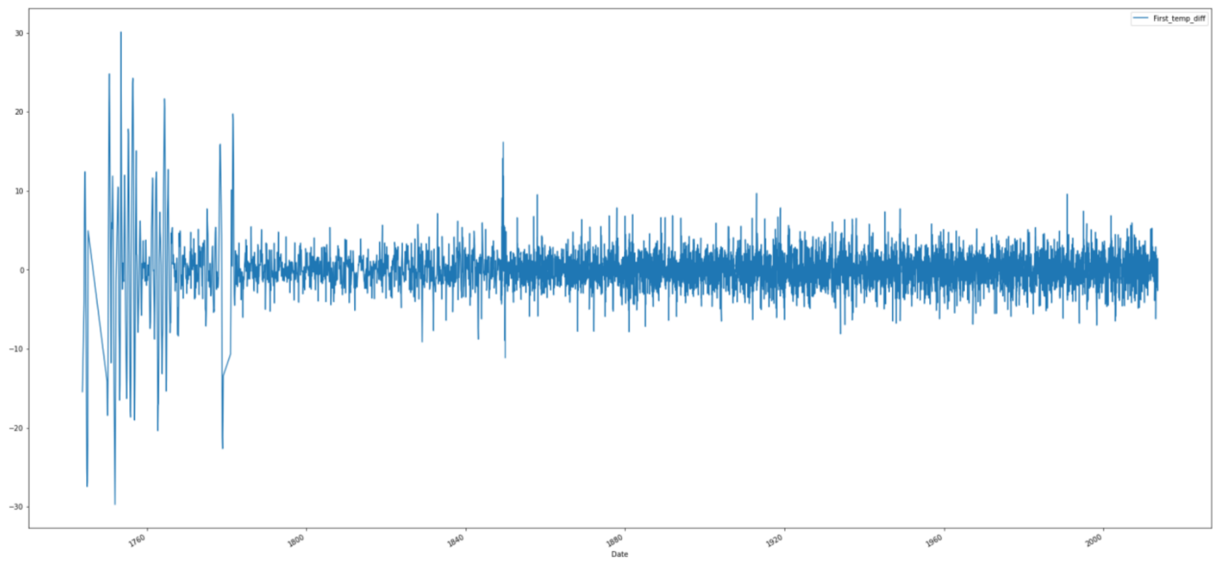
```
adfuller_test(df['First_temp_diff'].dropna())
```

```
ADF Test Statistic : -21.2396504049109
p-value : 0.0
#Lags Used : 35
Number of Observations Used : 7025
strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data has no unit root and is stationary
```

Plotting the data in the graphical format

```
df[['First_temp_diff']].plot(figsize=(30,15))
```

```
<AxesSubplot:xlabel='Date'>
```



Checking for the Seasonality.

Examine if there is a seasonality factor in data or not?

```
data2['month']=data2.index.month
```

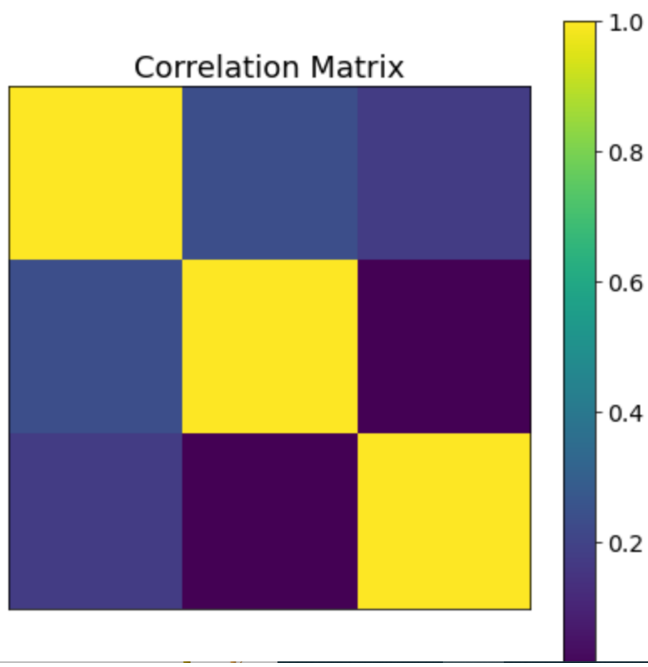
```
data2['year']=data2.index.year
```

```
data2.head()
```

58]:

Date	Temp	month	year
1849-01-01	8.819	1	1849
1849-02-01	9.577	2	1849
1849-03-01	11.814	3	1849
1849-04-01	13.704	4	1849
1849-05-01	14.834	5	1849

```
f = plt.figure(figsize=(7,7))
plt.matshow(data2.corr(),fignum=1)
plt.xticks([])
plt.yticks([])
cb = plt.colorbar()
cb.ax.tick_params(labelsize=14)
plt.title('Correlation Matrix', fontsize=18);
```



Pivot the Data to check Seasonality.

```python
pivot = data2.pivot_table(values='Temp',index='month',columns='year')
```

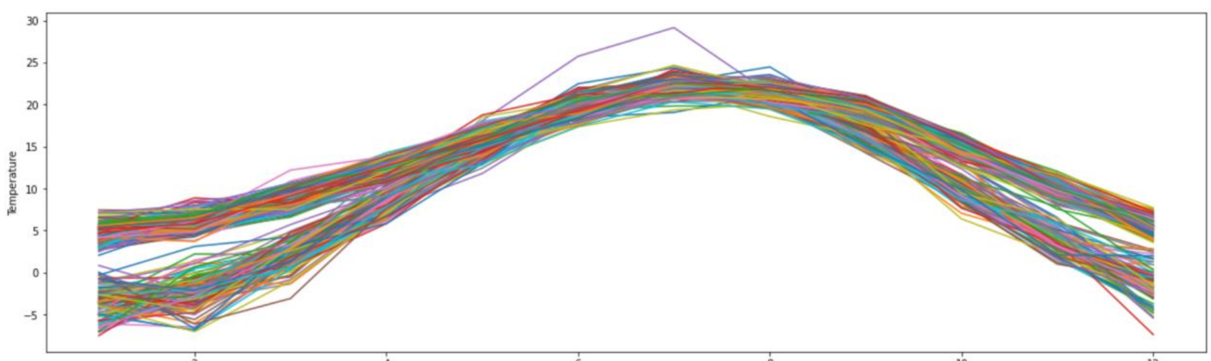
```python
pivot
```

year	1743	1744	1745	1750	1751	1752	1753	1754	1755	1756	...	2004	2005	2006	2007	2008	2009
month																	
1	NaN	NaN	-2.363	-4.310	-3.591	-7.588	-3.122	-2.252	-3.193	-1.985	...	4.080333	4.923000	7.135333	5.656000	5.216333	5.550333
2	NaN	NaN	-2.671	-2.719	-2.051	NaN	-1.467	-2.583	-1.802	0.259	...	6.100000	7.297000	7.547667	5.634000	6.754333	6.540667
3	NaN	NaN	1.363	2.773	3.256	3.322	4.207	2.728	1.112	NaN	...	12.184000	9.124667	7.527333	10.586000	9.733667	9.211000
4	NaN	9.788	8.209	8.848	7.992	7.402	8.099	NaN	8.714	NaN	...	13.806333	12.546000	12.051667	12.342000	13.113000	12.744000
5	NaN	15.708	NaN	15.411	NaN	NaN	15.330	NaN	15.238	NaN	...	17.817333	15.982333	17.123333	17.150000	15.800333	17.565667
6	NaN	21.210	NaN	19.017	20.724	NaN	20.820	20.075	19.964	20.488	...	19.872000	19.775333	21.395333	20.514333	21.474667	18.941667
7	NaN	22.207	NaN	24.203	22.668	NaN	22.524	22.503	NaN	22.452	...	22.246333	23.611667	24.655333	22.628333	23.020000	22.247667
8	NaN	NaN	NaN	22.135	21.547	NaN	21.324	21.461	NaN	21.208	...	22.101333	23.028333	21.920333	22.690000	22.305333	22.578000
9	NaN	14.922	NaN	17.445	15.812	NaN	15.548	16.281	16.137	17.345	...	20.669333	19.427667	19.365333	19.559000	20.686667	20.752667
10	NaN	8.968	NaN	9.076	NaN	9.391	10.479	11.477	8.669	9.662	...	14.205667	15.098000	14.368667	15.985667	15.699000	14.313333
11	3.264	3.161	NaN	NaN	NaN	5.831	3.363	NaN	3.599	2.894	...	9.758333	11.739667	11.861333	10.836333	11.423000	11.484000
12	NaN	-2.681	NaN	-1.093	NaN	-1.471	-2.854	-0.752	-2.381	-2.900	...	6.428000	6.582667	7.701667	5.474333	5.610667	5.425333

12 rows × 266 columns

```python
pivot.plot(figsize=(20,6))
plt.legend().remove()
plt.xlabel('Months')
plt.ylabel('Temperature')
```

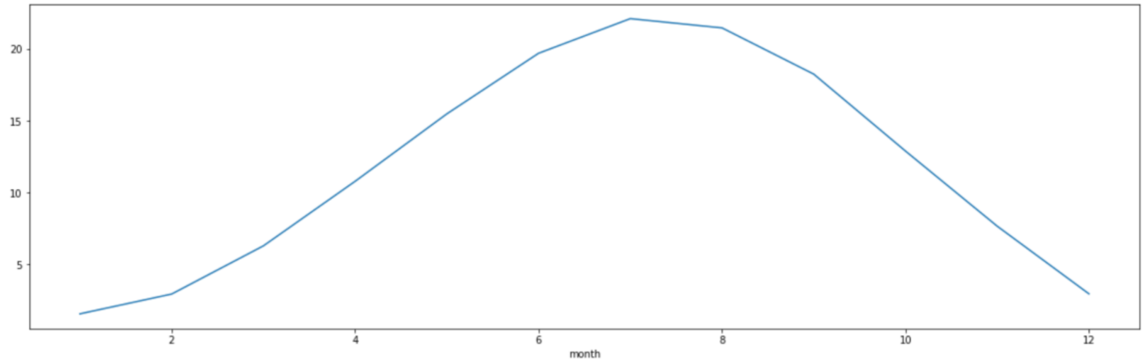
```
Text(0, 0.5, 'Temperature')
```



We found some kind of seasonality in the data.

]:
```
monthly_seasonality=pivot.mean(axis=1)
monthly_seasonality.plot(figsize=(20,6))
```

:[62]: <AxesSubplot:xlabel='month'>



Build time series

Using moving average

[63]:
```
df.head()
```

Out[63]:

	Temp	First_temp_diff
Date		
1849-01-01	8.819	NaN
1849-02-01	9.577	NaN
1849-03-01	11.814	NaN
1849-04-01	13.704	NaN
1849-05-01	14.834	NaN

```python
df=df[['First_temp_diff']]
df.dropna(inplace=True)
```

```python
df.head()
```

5]:

	First_temp_diff
Date	
1850-01-01	-1.732
1850-02-01	-1.002
1850-03-01	-1.449
1850-04-01	0.031
1850-05-01	1.799

```python
df['First_temp_diff'].rolling(window=5).mean()
```

```
Date
1850-01-01       NaN
1850-02-01       NaN
1850-03-01       NaN
1850-04-01       NaN
1850-05-01   -0.4706
               ...
2013-05-01    0.4336
2013-06-01    1.0236
2013-07-01    1.4060
2013-08-01    0.8454
2013-09-01    0.7614
Name: First_temp_diff, Length: 7061, dtype: float64
```

```python
value=pd.DataFrame(df['First_temp_diff'])
```

```python
temp_df=pd.concat([value,df['First_temp_diff'].rolling(window=5).mean()],axis=1)
```

```python
temp_df.columns=['actual_temp','forecast_temp']
temp_df.head()
```

;9]:

Date	actual_temp	forecast_temp
1850-01-01	-1.732	NaN
1850-02-01	-1.002	NaN
1850-03-01	-1.449	NaN
1850-04-01	0.031	NaN
1850-05-01	1.799	-0.4706

[70]:

```python
from sklearn.metrics import mean_squared_error,mean_absolute_percentage_error
np.sqrt(mean_squared_error(temp_df['forecast_temp'][4:],temp_df['actual_temp'][4:]))
```

Out[70]: 2.3934235122562058

ARIMA model

Using ARIMA

ARIMA stands for Autoregressive Integrated Moving Average. It is a combination of two models which are autoregressive and moving average.

ARIMA Model has three parameters:

p: it is the number of autoregressive lags.

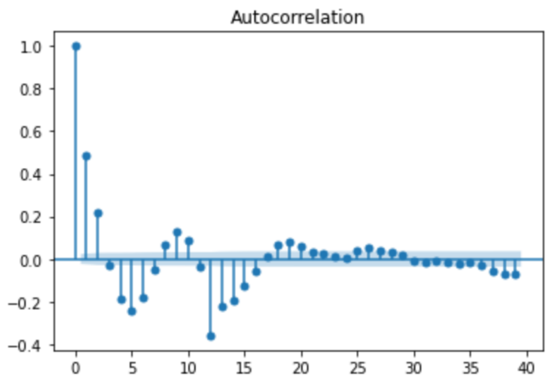
d: it is the order of differencing required to make the series stationary.

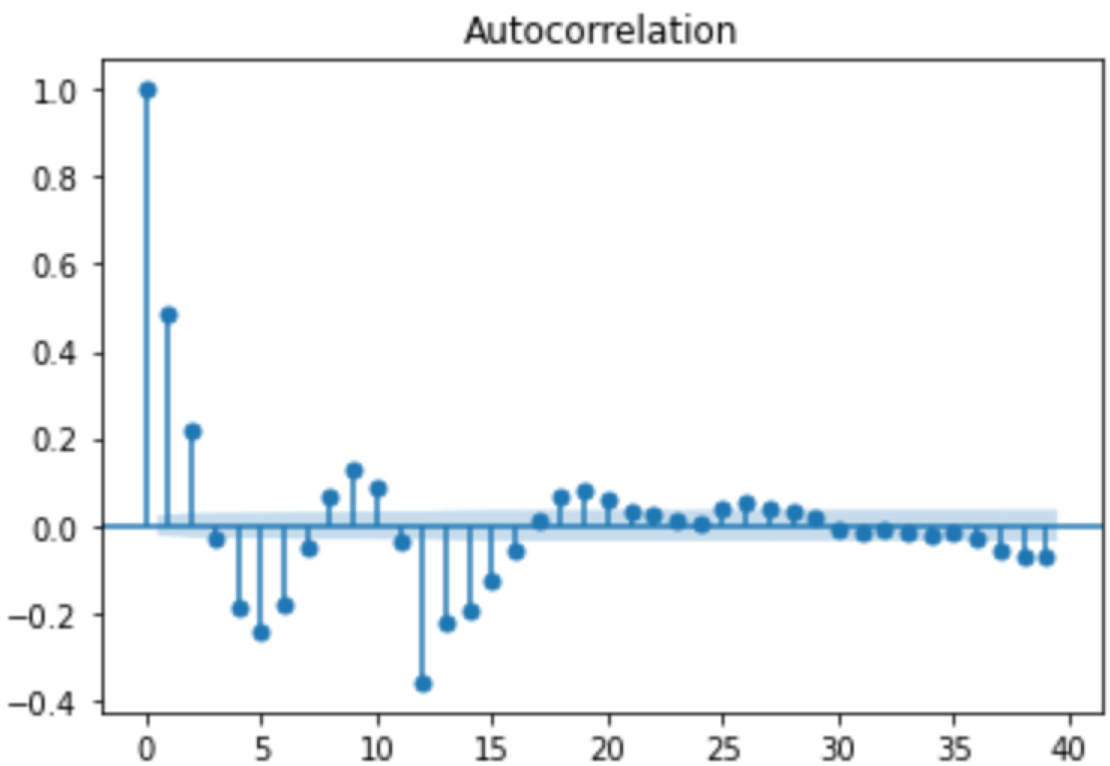
q: it is the number of moving average lags.

```
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
```

```
plot_acf(df['First_temp_diff'].dropna())
```

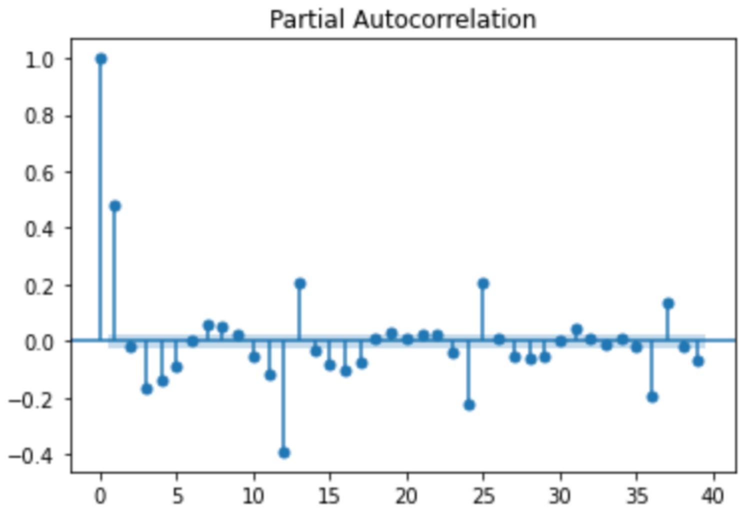
]:

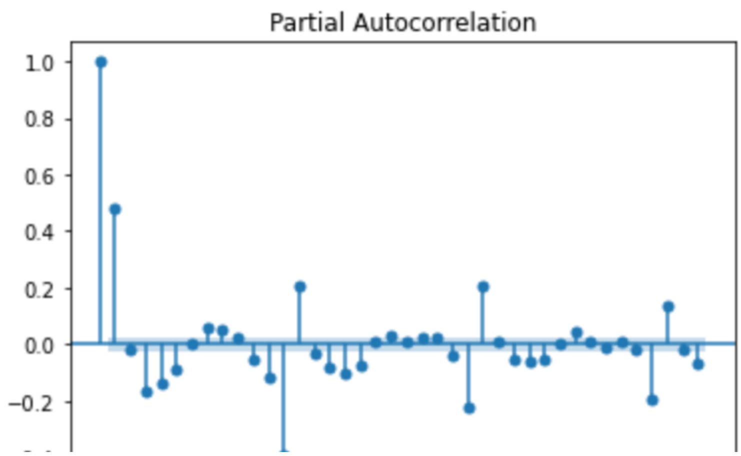




```
plot_pacf(df['First_temp_diff'].dropna())
```

]:





]:

```
df.isna().sum()
```

t[74]:

```
First_temp_diff    0
dtype: int64
```

]:

```
#training data
training_data=df[0:6000]
test_data = df[6000:]
```

```python
from statsmodels.tsa.arima_model import ARIMA
```

```python
arima = ARIMA(training_data,order=(2,1,3))
```

```
C:\Users\user\anaconda3\anaconda_install\lib\site-packages\statsmodels\tsa\base\tsa_model.py:581: ValueWarning:

A date index has been provided, but it has no associated frequency information and so will be ignored when e.g. foreca

C:\Users\user\anaconda3\anaconda_install\lib\site-packages\statsmodels\tsa\base\tsa_model.py:585: ValueWarning:

A date index has been provided, but it is not monotonic and so will be ignored when e.g. forecasting.

C:\Users\user\anaconda3\anaconda_install\lib\site-packages\statsmodels\tsa\base\tsa_model.py:581: ValueWarning:

A date index has been provided, but it has no associated frequency information and so will be ignored when e.g. foreca

C:\Users\user\anaconda3\anaconda_install\lib\site-packages\statsmodels\tsa\base\tsa_model.py:585: ValueWarning:

A date index has been provided, but it is not monotonic and so will be ignored when e.g. forecasting.
```

```python
#fit the model
model= arima.fit()
```

```python
#predictions
predictions=model.forecast(steps=len(test_data))[0]
predictions
```

```
79]: array([-0.19789817, -0.38474969, -0.35227303, ...,  0.0371716 ,
        0.03718145,  0.03719131])
```

```python
np.sqrt(mean_squared_error(test_data,predictions))
```

```
30]: 1.5120213784435974
```

So,We get a very less error in our prediction.