CHAPTER 1

INTRODUCTION

In this model, we are going to deal with the changes of temperature in global with the help of climate change prediction using time series analysis. Climate models, also known as general circulation models or GCMs, use mathematical equations to characterize how energy and matter interact in distinct parts of the ocean, atmosphere, and land.

1.1 WHAT IS MACHINE LEARNING

Machine learning is a branch of artificial intelligence (AI) and computer science which focuses on the use of data and algorithms to imitate the way that humans learn, gradually improving its accuracy.

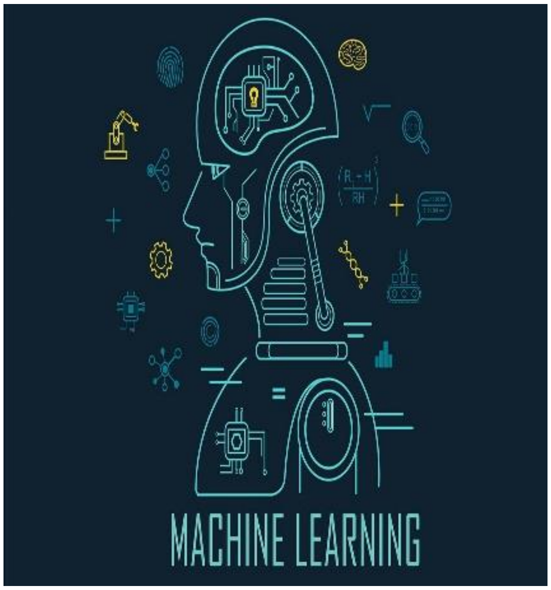


Fig.1.1 MACHINE LEARNING

1.2 WHY TIME SERIES?

Time series analysis is a specific way of analyzing a sequence of data points collected over an interval of time. In time series analysis, analysts record data points at consistent intervals over a set period rather than just recording the data points intermittently or randomly

1.It allows us to understand and compare things without losing the important, shared background of 'time'

2.It allows us to make forecasts

1.3 'MAKE-UP' OF A TIME SERIES

A time series is a set of repeated measurements of the same phenomenon, taken sequentially over time; it is thus an interesting variation of data type — it encapsulates this background of time, as well as…erm… anything else.

Time is (usually) the independent variable in a time series, whilst the dependent variable is the 'other thing.' It is useful to think of a time series as being made up of different components — this is known as decomposition modeling, and the resulting models can be additive or multiplicative in nature.

The four main components are:

1.Trend

2.Seasonality

3.Cyclicity

4.Irregularity

1.4 WHY CLIMATE CHANGE PREDICTION

To predict future climate, scientists use computer programs called climate models to understand how our planet is changing. Climate models work like a laboratory in a computer. They allow scientists to study how distinct factors interact to influence a region's climate.

1.5 WHAT IS MACHINE LEARNING PREDICTION

Prediction in machine learning refers to the output of an algorithm trained on a historical dataset. The algorithm then generates probable values for unknown variables in each record of the new data. The purpose of prediction in machine learning is to project a probable data set that relates back to the original data. Prediction is used to fit a shape as closely to the data as possible.

Prediction can be used to forecast the future and to predict the probability of an outcome. It can also be used to forecast future requirements or run a what-if analysis. One prediction tool is regression analysis used to determine the relationship between two variables.

Fig 1.5 MACHINE LEARNING PREDICTION

CHAPTER 2

SYSTEM ANALYSIS

2.1 AIM AND SCOPE

The main aim of this project is to develop a Climate Prediction Model Using time series analysis. Future changes are expected to include a warmer atmosphere, a warmer and more acidic ocean, higher sea levels, and larger changes in precipitation patterns. The extent of future climate change depends on what we do now to reduce greenhouse gas emissions. The more we emit, the larger future changes will be.

The scope of this project is to ensure high monitoring of climate and its temperature and to prevent climate problems such as Drought, sea level rise, heat waves, extreme weather, ocean acidities, etc....

2.2 METHODOLOY

To predict future climate, scientists use computer programs called climate models to understand how our planet is changing. Climate models work like a laboratory in a computer. They allow scientists to study how distinct factors interact to influence a region's climate. 3

The four main components are:

1. Trend

2. Seasonality

3. Cyclicity

4. Irregularity

2.2.1 TREND

Persistent over an extended period, the trend is the overall increase or decrease of the series during that time. See in the picture above how the series exhibits an upwards trend

2.2.2 SEASONALITY

Seasonality is the presence of variations that occur at specific regular intervals; it is the component of the data and series that experiences regular and predictable changes over a fixed period. Seasonality is illustrated in the picture above — notice the six identical 'up-down' fluctuations seen at regular intervals of x minutes. The peaks and troughs could also be illustrative of a seasonal component.

2.2.3 CYCLICITY

Cyclicity refers to the variation caused by circumstances, which repeat at irregular intervals. Seasonal behavior is very strictly regular, meaning there is a precise amount of time between the peaks and troughs of the data; cyclical behavior, on the other hand, can drift over time because the time between periods is not precise. For example, the stock market tends to cycle between periods of high and low values, but there is no set amount of time between those fluctuations. Cyclicity is illustrated in the picture above; in this case, the cyclicity seems to be due to specific events taking place before each occurrence.

2.2.4 IRREGULARITY

Irregularity is the unpredictable component of a time series — 'randomness.' This component cannot be explained by any other component and includes variations which occur due to unpredictable factors that do not repeat in set patterns. In the picture above; the magnifying glass illustrates this rough, random, and irregular component.

CHAPTER 3

ALGORITHM AND METHODS

3.1 GENERAL

Python is a high-level, general-purpose programming language. Its design philosophy emphasizes code readability with the use of significant indentation via the off-side rule. Python is dynamically typed, and garbage collected. It supports multiple programming paradigms, including structured (particularly procedural), object-oriented and functional programming. It is often described as a "batteries included" language due to its comprehensive standard library.

Django is a high-level Python web framework that enables rapid development of secure and maintainable websites. Built by experienced developers, Django takes care of much of the hassle of web development, so you can focus on writing your app without needing to waste a lot of time for no reason.

The HyperText Markup Language or HTML is the standard markup language for documents designed to be displayed in a web browser. It is often helped by technologies like Cascading Style Sheets and scripting languages like JavaScript.

3.2 ALGORITHM

3.2.1 PULLING THE DATA

This tutorial assumes that you are familiar with Jupyter notebooks and have at least some experiences with Pandas. In this section, we will start getting our feet wet by getting hold of some climate data and pulling it into our Jupyter Notebook with Pandas.

We will be using one dataset an estimate of global surface temperature change, from Kaggle

Firstly, download the datasets in CSV format (using the links above) and read them

in using pandas: Notice that in these cases, when reading in the datasets, we must skip several rows to get what we want — this is due to how the datasets are structured.

The temperature data represents temperature anomalies (differences from the mean/expected value) per month and per season (DJF=Dec-Feb, MAM=Mar-May, etc.). We will not be working with absolute temperature data as in climate change studies, anomalies are more important than absolute temperature. A positive anomaly indicates that the observed temperature was warmer than the baseline, while a negative anomaly indicates that it was cooler than the baseline.

3.2.2 WRANGLING

“Data wrangling is the process of transforming and mapping data from one “raw” data form into another format, to make it more valuable for downstream processes such as analytics.”

WRANGLING TEMPERATURE DATA

We will first wrangle Global temperature anomaly data. In doing so, we will look at several things:

·Using a DateTime index

·Basic manipulation and dealing with missing values

·Resampling to a different frequency.

SLICING AND SEARCHING

DateTime indexes make for convenient slicing of data, let us select all our data after the year 2011:

3.2.3 USEFUL FUNCTIONS

Pandas provides an entire range of other functions that can be especially useful when dealing with time series data — we cannot cover them all in this tutorial, but some are listed below:

- **DataFrame.rolling** → provides rolling window calculations
- **Pandas.to_datetime → a replacement for datetime.datetime's** *strptime function, it is more useful as it can infer the format*
- **TimSeries.shift & TimSeries.tshift** → allows for shifting or lagging of the values of a time series backward and forwards in time.

3.2.4 VISUALIZING

Now that we have our datasets nicely wrangled, let us look at how to plot them. We will be using two plotting libraries, namely:

- Matplotlib
- Plotly

3.2.5 TIME SERIES CORRELATION

Although it seems obvious that both series are trending upwards, what we would like to do here is determine whether the temperature change is as a result

3.3 PREREQUISITE

- Requirement of windows version 7 and latest versions.
- Runs in google chrome and Mozilla Firefox.
- Proper internet connection.

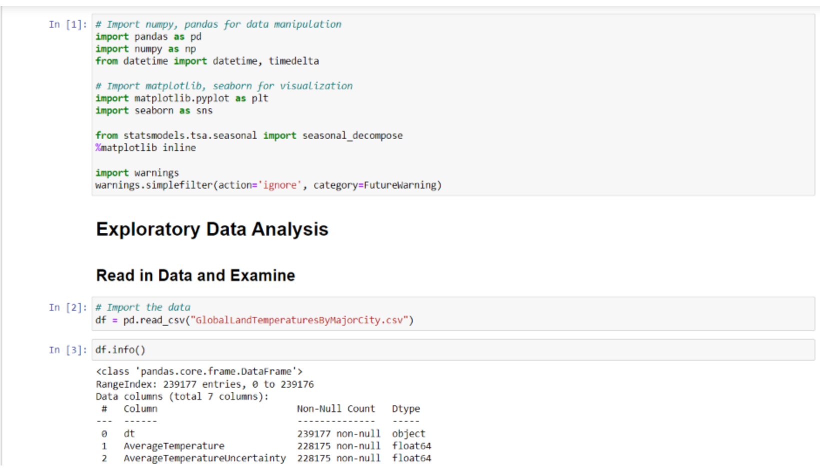
CHAPTER – 4

RESULTS & DISCUSSION

4.1 RESULTS

This Model was successfully developed and tested with different inputs, and we can get successful results. I have attached the output in the form of screen shots.

4.2 SOURCE CODE



```
In [1]: # Import numpy, pandas for data manipulation
import pandas as pd
import numpy as np
from datetime import datetime, timedelta

# Import matplotlib, seaborn for visualization
import matplotlib.pyplot as plt
import seaborn as sns

from statsmodels.tsa.seasonal import seasonal_decompose
%matplotlib inline

import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

Exploratory Data Analysis

Read in Data and Examine

```
In [2]: # Import the data
df = pd.read_csv("GlobalLandTemperaturesByMajorCity.csv")

In [3]: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 239177 entries, 0 to 239176
Data columns (total 7 columns):
 #   Column                         Non-Null Count   Dtype
---  ------                         --------------   -----
 0   dt                             239177 non-null  object
 1   AverageTemperature             228175 non-null  float64
 2   AverageTemperatureUncertainty  228175 non-null  float64
```

Fig 4.2.1 Source Code

In [4]:
```
df.head()
```
Out[4]:

	dt	AverageTemperature	AverageTemperatureUncertainty	City	Country	Latitude	Longitude
0	1849-01-01	26.704	1.435	Abidjan	Côte D'Ivoire	5.63N	3.23W
1	1849-02-01	27.434	1.362	Abidjan	Côte D'Ivoire	5.63N	3.23W
2	1849-03-01	28.101	1.612	Abidjan	Côte D'Ivoire	5.63N	3.23W
3	1849-04-01	26.140	1.387	Abidjan	Côte D'Ivoire	5.63N	3.23W
4	1849-05-01	25.427	1.200	Abidjan	Côte D'Ivoire	5.63N	3.23W

In [5]:
```
df.tail()
```
Out[5]:

	dt	AverageTemperature	AverageTemperatureUncertainty	City	Country	Latitude	Longitude
239172	2013-05-01	18.979	0.807	Xian	China	34.56N	108.97E
239173	2013-06-01	23.522	0.647	Xian	China	34.56N	108.97E
239174	2013-07-01	25.251	1.042	Xian	China	34.56N	108.97E
239175	2013-08-01	24.528	0.840	Xian	China	34.56N	108.97E
239176	2013-09-01	NaN	NaN	Xian	China	34.56N	108.97E

In [6]:
```
df.dtypes
```
Out[6]:
```
dt                               object
AverageTemperature              float64
AverageTemperatureUncertainty   float64
City                             object
Country                          object
Latitude                         object
Longitude                        object
dtype: object
```

Fig 4.2.2 Source Code

```
Longitude                        object
dtype: object
```

In [7]:
```
# Check the shape of the dataset
df.shape
```
Out[7]:
```
(239177, 7)
```

In [8]:
```
#checking for any null values
df.isnull().sum()
```
Out[8]:
```
dt                                   0
AverageTemperature               11002
AverageTemperatureUncertainty    11002
City                                 0
Country                              0
Latitude                             0
Longitude                            0
dtype: int64
```

In [9]:
```
df = df.dropna(how= 'any',axis=0)
```

In [10]:
```
df.shape
```
Out[10]:
```
(228175, 7)
```

In [11]:
```
df.rename(columns={'dt':'Date', 'AverageTemperature':'Avg_temp', 'AverageTemperatureUncertainty':'confidence_interval_temp'}, inp
df.head()
```
Out[11]:

	Date	Avg_temp	confidence_interval_temp	City	Country	Latitude	Longitude
0	1849-01-01	26.704	1.435	Abidjan	Côte D'Ivoire	5.63N	3.23W
1	1849-02-01	27.434	1.362	Abidjan	Côte D'Ivoire	5.63N	3.23W
2	1849-03-01	28.101	1.612	Abidjan	Côte D'Ivoire	5.63N	3.23W

Fig 4.2.3 Source Code

```
Out[10]: (228175, 7)
```

```
In [11]: df.rename(columns={'dt':'Date', 'AverageTemperature':'Avg_temp', 'AverageTemperatureUncertainty':'confidence_interval_temp'}, inp
         df.head()
```

Out[11]:

	Date	Avg_temp	confidence_interval_temp	City	Country	Latitude	Longitude
0	1849-01-01	26.704	1.435	Abidjan	Côte D'Ivoire	5.63N	3.23W
1	1849-02-01	27.434	1.362	Abidjan	Côte D'Ivoire	5.63N	3.23W
2	1849-03-01	28.101	1.612	Abidjan	Côte D'Ivoire	5.63N	3.23W
3	1849-04-01	26.140	1.387	Abidjan	Côte D'Ivoire	5.63N	3.23W
4	1849-05-01	25.427	1.200	Abidjan	Côte D'Ivoire	5.63N	3.23W

```
In [12]: df.head(100)
```

Out[12]:

	Date	Avg_temp	confidence_interval_temp	City	Country	Latitude	Longitude
0	1849-01-01	26.704	1.435	Abidjan	Côte D'Ivoire	5.63N	3.23W
1	1849-02-01	27.434	1.362	Abidjan	Côte D'Ivoire	5.63N	3.23W
2	1849-03-01	28.101	1.612	Abidjan	Côte D'Ivoire	5.63N	3.23W
3	1849-04-01	26.140	1.387	Abidjan	Côte D'Ivoire	5.63N	3.23W
4	1849-05-01	25.427	1.200	Abidjan	Côte D'Ivoire	5.63N	3.23W
...	...	...	...	...	...	...	...
151	1861-08-01	23.648	1.253	Abidjan	Côte D'Ivoire	5.63N	3.23W
154	1861-11-01	25.762	1.476	Abidjan	Côte D'Ivoire	5.63N	3.23W
155	1861-12-01	25.856	1.656	Abidjan	Côte D'Ivoire	5.63N	3.23W
156	1862-01-01	25.427	1.396	Abidjan	Côte D'Ivoire	5.63N	3.23W

Fig 4.2.4 Source Code

```
In [13]: df['Date'] = pd.to_datetime(df['Date'])
         df.set_index('Date', inplace=True)
         df.index

Out[13]: DatetimeIndex(['1849-01-01', '1849-02-01', '1849-03-01', '1849-04-01',
                        '1849-05-01', '1849-06-01', '1849-07-01', '1849-08-01',
                        '1849-09-01', '1849-10-01',
                        ...
                        '2012-11-01', '2012-12-01', '2013-01-01', '2013-02-01',
                        '2013-03-01', '2013-04-01', '2013-05-01', '2013-06-01',
                        '2013-07-01', '2013-08-01'],
                       dtype='datetime64[ns]', name='Date', length=228175, freq=None)
```

In order to get more information, Python Pandas library provides a 'describe' function to show the count, mean, standard deviation, min/ max value and the quantiles of our dataset:

```
In [14]: df.describe()
```

Out[14]:

	Avg_temp	confidence_interval_temp
count	228175.000000	228175.000000
mean	18.125969	0.969343
std	10.024800	0.979644
min	-26.772000	0.040000
25%	12.710000	0.340000
50%	20.428000	0.592000
75%	25.918000	1.320000
max	38.283000	14.037000

```
In [15]: df['Year'] = df.index.year
         df.head()
```

Fig 4.2.5 Source Code

max	38.283000	14.037000

```
In [15]: df['Year'] = df.index.year
         df.head()
```

Out[15]:

Date	Avg_temp	confidence_interval_temp	City	Country	Latitude	Longitude	Year
1849-01-01	26.704	1.435	Abidjan	Côte D'Ivoire	5.63N	3.23W	1849
1849-02-01	27.434	1.362	Abidjan	Côte D'Ivoire	5.63N	3.23W	1849
1849-03-01	28.101	1.612	Abidjan	Côte D'Ivoire	5.63N	3.23W	1849
1849-04-01	26.140	1.387	Abidjan	Côte D'Ivoire	5.63N	3.23W	1849
1849-05-01	25.427	1.200	Abidjan	Côte D'Ivoire	5.63N	3.23W	1849

```
In [16]: df.head(100)
```

Out[16]:

Date	Avg_temp	confidence_interval_temp	City	Country	Latitude	Longitude	Year
1849-01-01	26.704	1.435	Abidjan	Côte D'Ivoire	5.63N	3.23W	1849
1849-02-01	27.434	1.362	Abidjan	Côte D'Ivoire	5.63N	3.23W	1849
1849-03-01	28.101	1.612	Abidjan	Côte D'Ivoire	5.63N	3.23W	1849
1849-04-01	26.140	1.387	Abidjan	Côte D'Ivoire	5.63N	3.23W	1849
1849-05-01	25.427	1.200	Abidjan	Côte D'Ivoire	5.63N	3.23W	1849
...	...	...	...	...	...	...	...
1861-08-01	23.648	1.253	Abidjan	Côte D'Ivoire	5.63N	3.23W	1861
1861-11-01	25.762	1.476	Abidjan	Côte D'Ivoire	5.63N	3.23W	1861
1861-12-01	25.856	1.656	Abidjan	Côte D'Ivoire	5.63N	3.23W	1861

Fig 4.2.6 Source Code

```
In [17]: df.describe()
```

Out[17]:

	Avg_temp	confidence_interval_temp	Year
count	228175.000000	228175.000000	228175.000000
mean	18.125969	0.969343	1913.893209
std	10.024800	0.979644	62.025981
min	-26.772000	0.040000	1743.000000
25%	12.710000	0.340000	1869.000000
50%	20.428000	0.592000	1918.000000
75%	25.918000	1.320000	1966.000000
max	38.283000	14.037000	2013.000000

```
In [18]: df.plot(figsize=(15,5))
```

Out[18]: <AxesSubplot:xlabel='Date'>

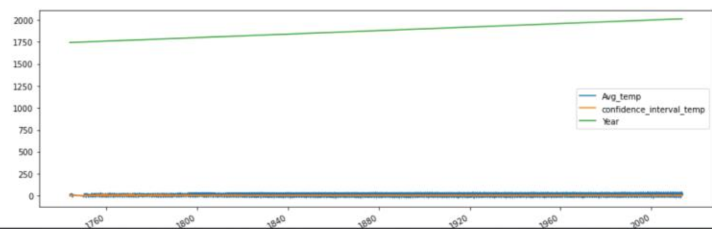


Fig 4.2.7 Source Code

```
In [19]: # Select the subset data from 2005 to 2010
         latest_df = df.loc['2005':'2010']
         # Inspect first 5 rows of the data
         latest_df.head(100)
```

Out[19]:

Date	Avg_temp	confidence_interval_temp	City	Country	Latitude	Longitude	Year
2005-01-01	26.715	0.426	Abidjan	Côte D'Ivoire	5.63N	3.23W	2005
2005-02-01	29.405	0.360	Abidjan	Côte D'Ivoire	5.63N	3.23W	2005
2005-03-01	29.246	0.268	Abidjan	Côte D'Ivoire	5.63N	3.23W	2005
2005-04-01	28.508	0.170	Abidjan	Côte D'Ivoire	5.63N	3.23W	2005
2005-05-01	27.416	0.287	Abidjan	Côte D'Ivoire	5.63N	3.23W	2005
...	...	...	...	...	...	...	...
2006-12-01	17.285	0.938	Addis Abeba	Ethiopia	8.84N	38.11E	2006
2007-01-01	18.484	0.347	Addis Abeba	Ethiopia	8.84N	38.11E	2007
2007-02-01	18.959	0.520	Addis Abeba	Ethiopia	8.84N	38.11E	2007
2007-03-01	20.045	0.648	Addis Abeba	Ethiopia	8.84N	38.11E	2007
2007-04-01	19.619	0.422	Addis Abeba	Ethiopia	8.84N	38.11E	2007

100 rows × 7 columns

For simplicity, the latest_df column above shows the average temperatures for the first 5 months of 2005 and last 5 months of 2007 according to the dataset.

```
In [20]: latest_df[['Country','Avg_temp']].groupby(['Country']).mean().sort_values('Avg_temp')
```

Out[20]:

Country	Avg_temp

Fig 4.2.8 Source Code

Country	Avg_temp
Spain	12.348403
United States	12.864532

```
In [21]: resample_df = latest_df[['Avg_temp']].resample('A').mean()

In [22]: resample_df.head()
```

Out[22]:

Date	Avg_temp
2005-12-31	19.607239
2006-12-31	19.793993
2007-12-31	19.854270
2008-12-31	19.608778
2009-12-31	19.833752

Data Visualization

Let's explore this time series as a data visualization:

```
In [23]: resample_df.plot(title='Temperature Changes from 2005-2010',figsize=(8,5))
         plt.ylabel('Temperature',fontsize=12)
         plt.xlabel('Year',fontsize=12)
         plt.legend()

Out[23]: <matplotlib.legend.Legend at 0x1980dc21be0>
```

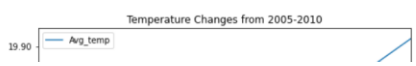
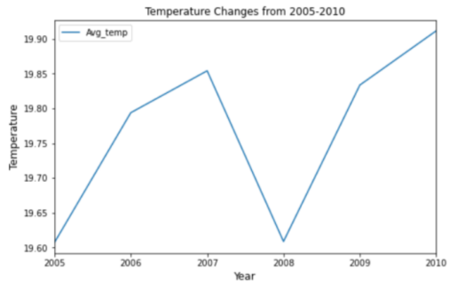


Fig 4.2.9 Source Code

```
Out[23]: <matplotlib.legend.Legend at 0x1980dc21be0>
```




```
In [24]: from statsmodels.tsa.stattools import adfuller

         print('Dickey Fuller Test Results:')
         test_df = adfuller(resample_df.iloc[:,0].values, autolag='AIC')
         df_output = pd.Series(test_df[0:4], index=['Test Statistic','p-value','Lags Used','Numberr of Observations Used'])
         for key, value in test_df[4].items():
             df_output['Critical Value (%s)'%key] = value
         print(df_output)

         Dickey Fuller Test Results:
         Test Statistic                 -4.115335
         p-value                         0.000914
         Lags Used                       1.000000
         Numberr of Observations Used    4.000000
```

Fig 4.2.10 Source Code

```
In [25]: decomp = seasonal_decompose(resample_df,freq=3)

         trend = decomp.trend
         seasonal = decomp.seasonal
         residual = decomp.resid

In [26]: #Plotting the Original Time Series
         plt.subplot(411)
         plt.plot(resample_df)
         plt.xlabel('Original')
         plt.figure(figsize=(6,5))

         #Plotting the Trend Component
         plt.subplot(412)
         plt.plot(trend)
         plt.xlabel('Trend')
         plt.figure(figsize=(8,5))

         #Plotting the Seasonal Component
         plt.subplot(413)
         plt.plot(seasonal)
         plt.xlabel('Seasonal')
         plt.figure(figsize=(7,5))

         #Plotting the Residual Component
         plt.subplot(414)
         plt.plot(residual)
         plt.xlabel('Residual')
         plt.figure(figsize=(9,5))

         plt.tight_layout()
```

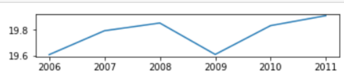
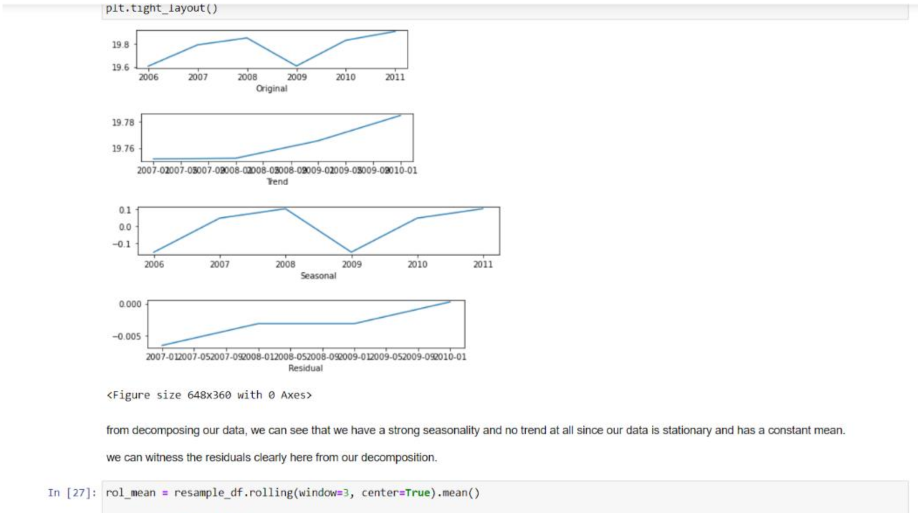


Fig 4.2.11 Source Code



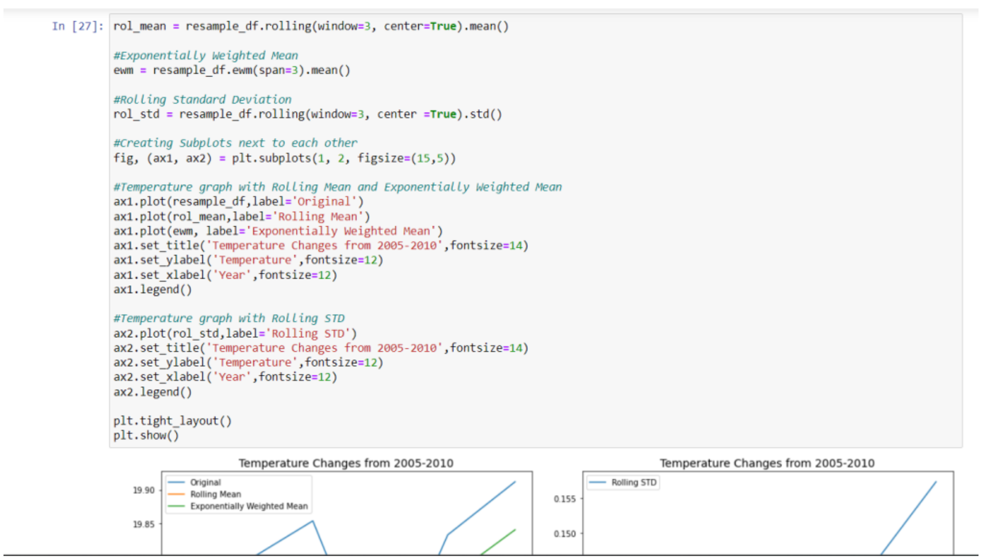

```
<Figure size 648x360 with 0 Axes>
```

from decomposing our data, we can see that we have a strong seasonality and no trend at all since our data is stationary and has a constant mean.

we can witness the residuals clearly here from our decomposition.

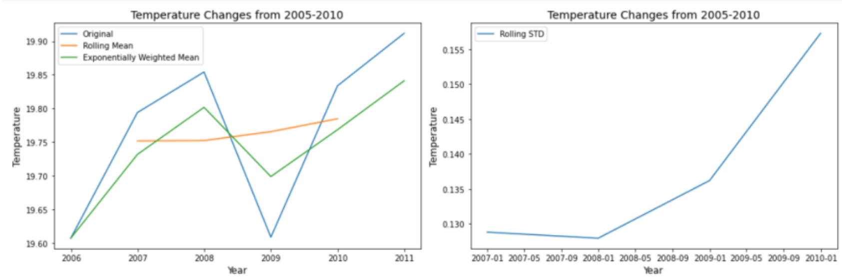
```
In [27]: rol_mean = resample_df.rolling(window=3, center=True).mean()
```

Fig 4.2.12 Source Code



```
In [27]: rol_mean = resample_df.rolling(window=3, center=True).mean()

#Exponentially Weighted Mean
ewm = resample_df.ewm(span=3).mean()

#Rolling Standard Deviation
rol_std = resample_df.rolling(window=3, center =True).std()

#Creating Subplots next to each other
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15,5))

#Temperature graph with Rolling Mean and Exponentially Weighted Mean
ax1.plot(resample_df,label='Original')
ax1.plot(rol_mean,label='Rolling Mean')
ax1.plot(ewm, label='Exponentially Weighted Mean')
ax1.set_title('Temperature Changes from 2005-2010',fontsize=14)
ax1.set_ylabel('Temperature',fontsize=12)
ax1.set_xlabel('Year',fontsize=12)
ax1.legend()

#Temperature graph with Rolling STD
ax2.plot(rol_std,label='Rolling STD')
ax2.set_title('Temperature Changes from 2005-2010',fontsize=14)
ax2.set_ylabel('Temperature',fontsize=12)
ax2.set_xlabel('Year',fontsize=12)
ax2.legend()

plt.tight_layout()
plt.show()
```


Fig 4.2.13 Source Code



In [28]:
```
rol_mean.dropna(inplace=True)
ewm.dropna(inplace=True)

print ('Dickey-Fuller Test for the Rolling Mean:')
df_test = adfuller(rol_mean.iloc[:,0].values, autolag='AIC')
df_output = pd.Series(df_test[0:4],index=['Test Statistics','p-value','Lags Used','Number of Observations Used'])
for key,value in df_test[4].items():
    df_output['Critical Value (%s)'%key] = value
print(df_output)
print('')
print('Dickey-Fuller Test for the Exponentially Weighted Mean:')
df_test = adfuller(ewm.iloc[:,0].values, autolag='AIC')
df_output = pd.Series(df_test[0:4], index=['Test Statistic','p-value','Lags Used','Number of Observations Used'])
for key,value in df_test[4].items():
    df_output['Critical Value (%s)'%key] = value
print(df_output)
```

Fig 4.2.14 Source Code

```
dtype: float64
```

In [29]:
```
diff_rol_mean = resample_df = rol_mean
diff_rol_mean.dropna(inplace=True)
diff_rol_mean.head()
```

Out[29]:

Date	Avg_temp
2006-12-31	19.751834
2007-12-31	19.752347
2008-12-31	19.765600
2009-12-31	19.784755

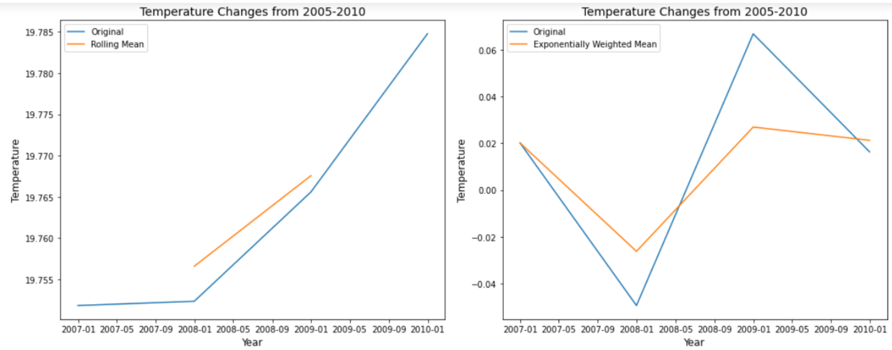
In [30]:
```
diff_ewm = resample_df - ewm
diff_ewm.dropna(inplace=True)
diff_ewm.head()
```

Out[30]:

Date	Avg_temp
2006-12-31	0.020092
2007-12-31	-0.049411
2008-12-31	0.066765
2009-12-31	0.016285

In [31]:
```
df_rol_mean_diff = diff_rol_mean.rolling(window=3, center=True).mean()

#Exponentially Weighted Mean of the difference
df_ewm_diff = diff_ewm.ewm(span=3).mean()
```

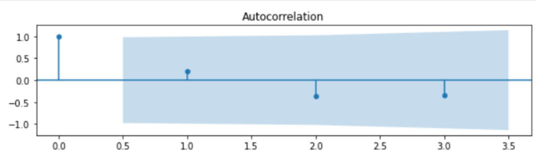
Fig 4.2.15 Source Code



In [32]:
```
print ('Dickey-Fuller Test for the Difference between the Original and Rolling Mean:')
dftest = adfuller(diff_rol_mean.iloc[:,0].values, autolag='AIC')
dfoutput = pd.Series(dftest[0:4], index=['Test Statistc','p-value','Lags Used','Number of Observations Used'])
for key,value in dftest[4].items():
    dfoutput['Critical Value (%s)'%key] = value
print(dfoutput)
print('')
print('Dickey-Fuller Test for the Difference between the Original and Exponentially Weighted Mean:')
dftest = adfuller(diff_ewm.iloc[:,0].values, autolag='AIC')
dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-values','Lags Used','Number of Observations Used'])
for key, value in dftest[4].items():
    dfoutput['Critical Value (%s)'%key] = value
print(dfoutput)
```

Fig 4.2.16 Source Code

In [33]:
```
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.graphics.tsaplots import plot_pacf
from matplotlib import pyplot

#plot the Autocorrelation graph
pyplot.figure(figsize=(10,5))
pyplot.subplot(211)
plot_acf(resample_df, ax=pyplot.gca())
pyplot.show()
```



In [34]:
```
# Plot the Partial Autocorrelation graph
import statsmodels.api as sm

sm.graphics.tsa.plot_pacf(resample_df.values.squeeze(), lags=1, method="ywm")
plt.figure(figsize=(8,4))
plt.show()
```

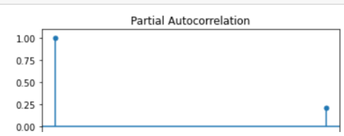


Fig 4.2.17 Source Code

CHAPTER 5

CONCLUSION

In this paper, The Climate Change prediction using time series analysis is presented the system use series of data has been stored in the form of dataset in Kaggle website.

BEWARE OF TREND

Trends occur in many time series, and before embarking on an exploration of the relationship between two different time series, you should first attempt to measure and control this trend. In doing so, you will lessen the chance of encountering spurious correlations. But even de-trending a time series cannot protect you from all spurious correlations — patterns such as seasonality, periodicity and autocorrelation can too.

BE AWARE OF HOW YOU DEAL WITH A TREND

It is possible to de-trend naively. Attempting to achieve stationarity using (for example) a first differences approach may spoil your data if you are looking for lagged effects.

REFERENCE

https://www.kaggle.com/datasets/berkeleyearth/climate-change-earth-surface-temperature-data?select=GlobalLandTemperaturesByCountry.csv

https://github.com/Arunfi143/Our_Project_repo

https://towardsdatascience.com/time-series-analysis-and-climate-change-7bb4371021e