Data analysis on India Weather Data

Introduction:

This notebook is for analysing the data on India weather, the data was gathered from https://data.gov.in/

Dataset description:

The data contains information about the average temperature(Celsius) that was recorded in india per month over the years 1901-2017.

Reason for obtaining this dataset

Due to me requiring to gather more insight on my main dataset on crop production in india, I wanted to see if the weather in Inida has any effect on how much crops are produced in India.

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Getting the average weather of india for each month per year

In [2]:
```
df = pd.read_csv('Weather Data in India from 1901 to 2017.csv')
df
```

Out[2]:

	Unnamed: 0	YEAR	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC
0	0	1901	17.99	19.43	23.49	26.41	28.28	28.60	27.49	26.98	26.26	25.08	21.73	18.95
1	1	1902	19.00	20.39	24.10	26.54	28.68	28.44	27.29	27.05	25.95	24.37	21.33	18.78
2	2	1903	18.32	19.79	22.46	26.03	27.93	28.41	28.04	26.63	26.34	24.57	20.96	18.29
3	3	1904	17.77	19.39	22.95	26.73	27.83	27.85	26.84	26.73	25.84	24.36	21.07	18.84
4	4	1905	17.40	17.79	21.78	24.84	28.32	28.69	27.67	27.47	26.29	26.16	22.07	18.71
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
112	112	2013	18.88	21.07	24.53	26.97	29.06	28.24	27.50	27.22	26.87	25.63	22.18	19.69
113	113	2014	18.81	20.35	23.34	26.91	28.45	29.42	28.07	27.42	26.61	25.38	22.53	19.50
114	114	2015	19.02	21.23	23.52	26.52	28.82	28.15	28.03	27.64	27.04	25.82	22.95	20.21
115	115	2016	20.92	23.58	26.61	29.56	30.41	29.70	28.18	28.17	27.72	26.81	23.90	21.89
116	116	2017	20.59	23.08	25.58	29.17	30.47	29.44	28.31	28.12	28.11	27.24	23.92	21.47

117 rows × 14 columns

Dropping unnamed column

After importing the dataset I realised there is a column called unnamedand this column has no useful data so I will be removing it

In [3]:
```
df.drop(columns = 'Unnamed: 0', axis = 0, inplace = True)
```

In [4]:
```
df
```

Out[4]:

	YEAR	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC
0	1901	17.99	19.43	23.49	26.41	28.28	28.60	27.49	26.98	26.26	25.08	21.73	18.95
1	1902	19.00	20.39	24.10	26.54	28.68	28.44	27.29	27.05	25.95	24.37	21.33	18.78
2	1903	18.32	19.79	22.46	26.03	27.93	28.41	28.04	26.63	26.34	24.57	20.96	18.29
3	1904	17.77	19.39	22.95	26.73	27.83	27.85	26.84	26.73	25.84	24.36	21.07	18.84
4	1905	17.40	17.79	21.78	24.84	28.32	28.69	27.67	27.47	26.29	26.16	22.07	18.71
...	...	...	...	...	...	...	...	...	...	...	...	...	...
112	2013	18.88	21.07	24.53	26.97	29.06	28.24	27.50	27.22	26.87	25.63	22.18	19.69
113	2014	18.81	20.35	23.34	26.91	28.45	29.42	28.07	27.42	26.61	25.38	22.53	19.50
114	2015	19.02	21.23	23.52	26.52	28.82	28.15	28.03	27.64	27.04	25.82	22.95	20.21
115	2016	20.92	23.58	26.61	29.56	30.41	29.70	28.18	28.17	27.72	26.81	23.90	21.89
116	2017	20.59	23.08	25.58	29.17	30.47	29.44	28.31	28.12	28.11	27.24	23.92	21.47

117 rows × 13 columns

For my dataset of India cropping production the data is gathered based on the years 1997-2015 therefore I will be filtering this weather to 1997-2015

In [5]:
```
df.dtypes
```

Out[5]:
```
YEAR      int64
JAN     float64
FEB     float64
MAR     float64
APR     float64
MAY     float64
JUN     float64
JUL     float64
AUG     float64
SEP     float64
OCT     float64
NOV     float64
DEC     float64
dtype: object
```

Rearranging weather dataset

I decided to rearrange the intial dataset, this is because I would prefer having the month as values rather than column names,
Therefore making the data is more readable

In [6]:
```
#creating a new array to contain the dates
dates = {}

i = 0
# for each of the years in the dataset
for year in df['YEAR']:
    #for each of the column names, which are the months
    for month in df.columns[1:]:
        #convert the values to strings and then add the m/y
```

```
        dat = str(month) + '/' + str(year)

        #assigning the dat value in the array the value which corresponds to each month ir
        dates[dat] = df[month][i]
    i += 1

#converting array to a dataframe
weather_df = pd.DataFrame(pd.Series(dates).reset_index())

weather_df.columns = ['Date', 'Temp']

#converting date column to datetime
weather_df['Date'] = pd.to_datetime(weather_df['Date'], format= '%b/%Y')

#splitting creating year and month column of date column
weather_df['Year'] = weather_df['Date'].dt.year
weather_df['Month'] = weather_df['Date'].dt.month_name()

weather_df
```

Out[6]:

	Date	Temp	Year	Month
0	1901-01-01	17.99	1901	January
1	1901-02-01	19.43	1901	February
2	1901-03-01	23.49	1901	March
3	1901-04-01	26.41	1901	April
4	1901-05-01	28.28	1901	May
...	...	...	...	...
1399	2017-08-01	28.12	2017	August
1400	2017-09-01	28.11	2017	September
1401	2017-10-01	27.24	2017	October
1402	2017-11-01	23.92	2017	November
1403	2017-12-01	21.47	2017	December

1404 rows × 4 columns

After imprving the overall readbility of the dataset, I will proceed in filtering the dataset to contain only weather data on the years 1997 - 2015 because this is the tiem period my main dataset on crop production has

In [7]:

```
weather_df = weather_df[(weather_df['Year'] >= 1997) & (weather_df['Year'] < 2016)]
weather_df
```

Out[7]:

	Date	Temp	Year	Month
1152	1997-01-01	17.86	1997	January
1153	1997-02-01	19.88	1997	February
1154	1997-03-01	23.64	1997	March
1155	1997-04-01	25.55	1997	April
1156	1997-05-01	27.86	1997	May
...	...	...	...	...

	Date	Temp	Year	Month
1375	2015-08-01	27.64	2015	August
1376	2015-09-01	27.04	2015	September
1377	2015-10-01	25.82	2015	October
1378	2015-11-01	22.95	2015	November
1379	2015-12-01	20.21	2015	December

228 rows × 4 columns

Creating season column from the month column

My next task was to create a seperate column which mapps each month with the season that month is associated to

This alligns with the data entries in my main dataset due to that having data on crop production per season

In [8]:
```
seasons = {'January': 'Winter', 'February':'Winter', 'March': 'Winter', 'April': 'Summer',
                  'June': 'Summer', 'July':'Autumn', 'August': 'Autumn', 'September': 'Aut
                  'November':'Winter', 'December':'Winter'}


weather_df['Season'] = weather_df['Month'].apply(lambda x: seasons[x])
weather_df
```

```
C:\Users\mikos\AppData\Local\Temp/ipykernel_23712/3633730968.py:6: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_gu
ide/indexing.html#returning-a-view-versus-a-copy
  weather_df['Season'] = weather_df['Month'].apply(lambda x: seasons[x])
```

Out[8]:

	Date	Temp	Year	Month	Season
1152	1997-01-01	17.86	1997	January	Winter
1153	1997-02-01	19.88	1997	February	Winter
1154	1997-03-01	23.64	1997	March	Winter
1155	1997-04-01	25.55	1997	April	Summer
1156	1997-05-01	27.86	1997	May	Summer
...	...	...	...	...	...
1375	2015-08-01	27.64	2015	August	Autumn
1376	2015-09-01	27.04	2015	September	Autumn
1377	2015-10-01	25.82	2015	October	Autumn
1378	2015-11-01	22.95	2015	November	Winter
1379	2015-12-01	20.21	2015	December	Winter

228 rows × 5 columns

Data check

Searching for null values in the dataset

My next step after some data transformation was done was to to check if the dataset is clean or not.
To do this i began to search if there are any null values

In [9]:
```
weather_df.isna().sum()
```

Out[9]:
```
Date      0
Temp      0
Year      0
Month     0
Season    0
dtype: int64
```

There are no null values

Searching for duplciated rows

Step 2:
The next step taking to check if my data is clean or not was ot check if there are any duplciated values

In [10]:
```
duplicated_rows = weather_df[weather_df.duplicated()]
```

In [11]:
```
duplicated_rows
```

Out[11]:

Date	Temp	Year	Month	Season

Conclusion:

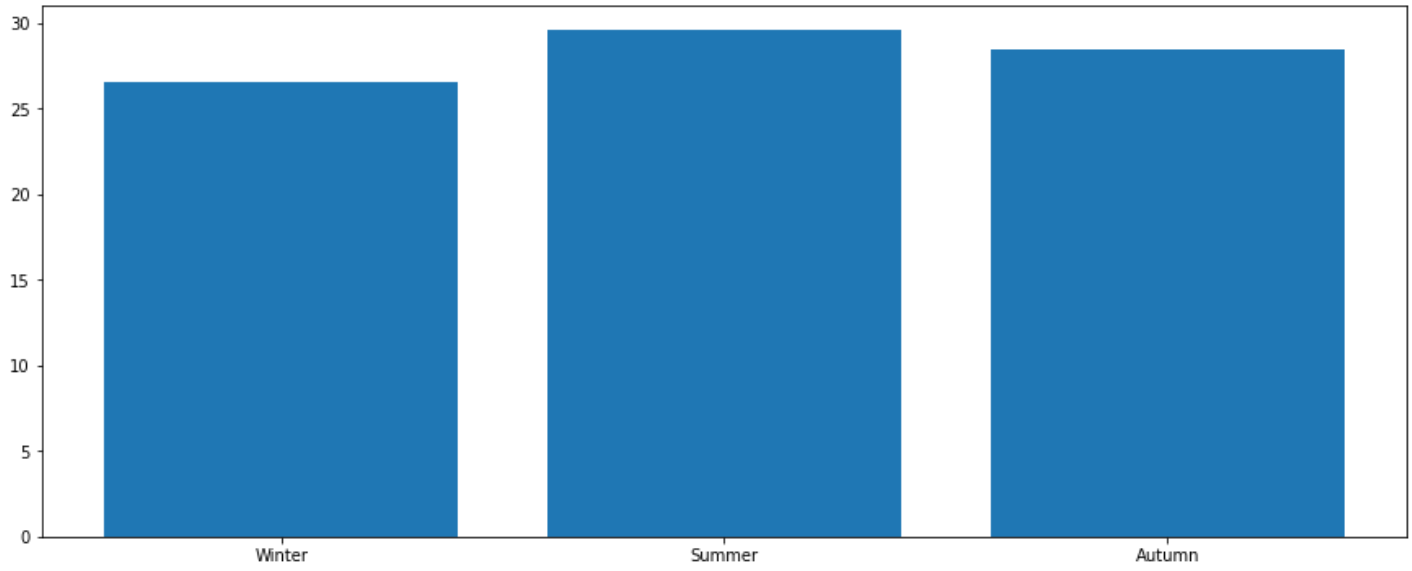
I can conclude that this dataset is clean but mainly needed data transformation

EDA

The first step in my EDA since I am dealing with data based on time(Time series data), I decided to check if the dataset is seasonal or not.
Since in my main dataset I would like to use seasons as a factor that also determines how mcuh will be porduced I would like to see if the indian weather correlates with this (Season - Temp)

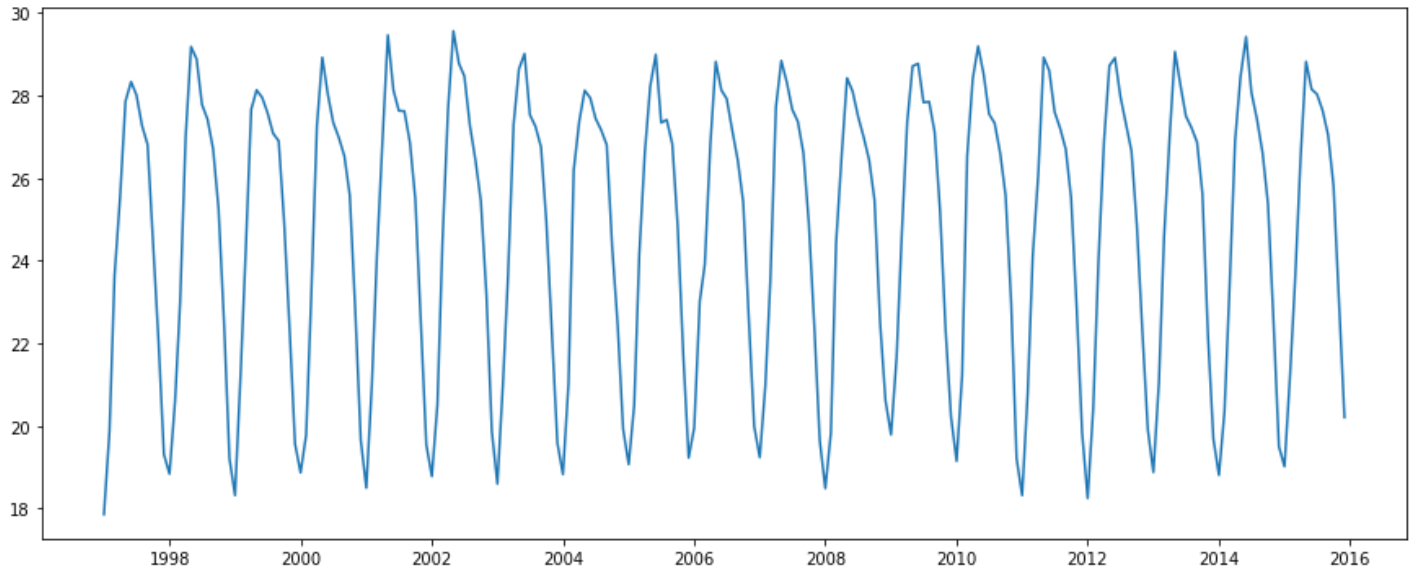
Checking if the data is seasonal or not

In [12]:
```
fig, ax = plt.subplots(figsize=(15,6))

ax.bar(weather_df['Season'], weather_df['Temp'])
plt.show()
```



In [13]:

```
fig, ax = plt.subplots(figsize=(15,6))

ax.plot(weather_df['Date'], weather_df['Temp'])
plt.show()
```



Conclusion:

I can conclude that this dataset is seasonal, this due to the high spikes and sudden low spikes shown in the graph.

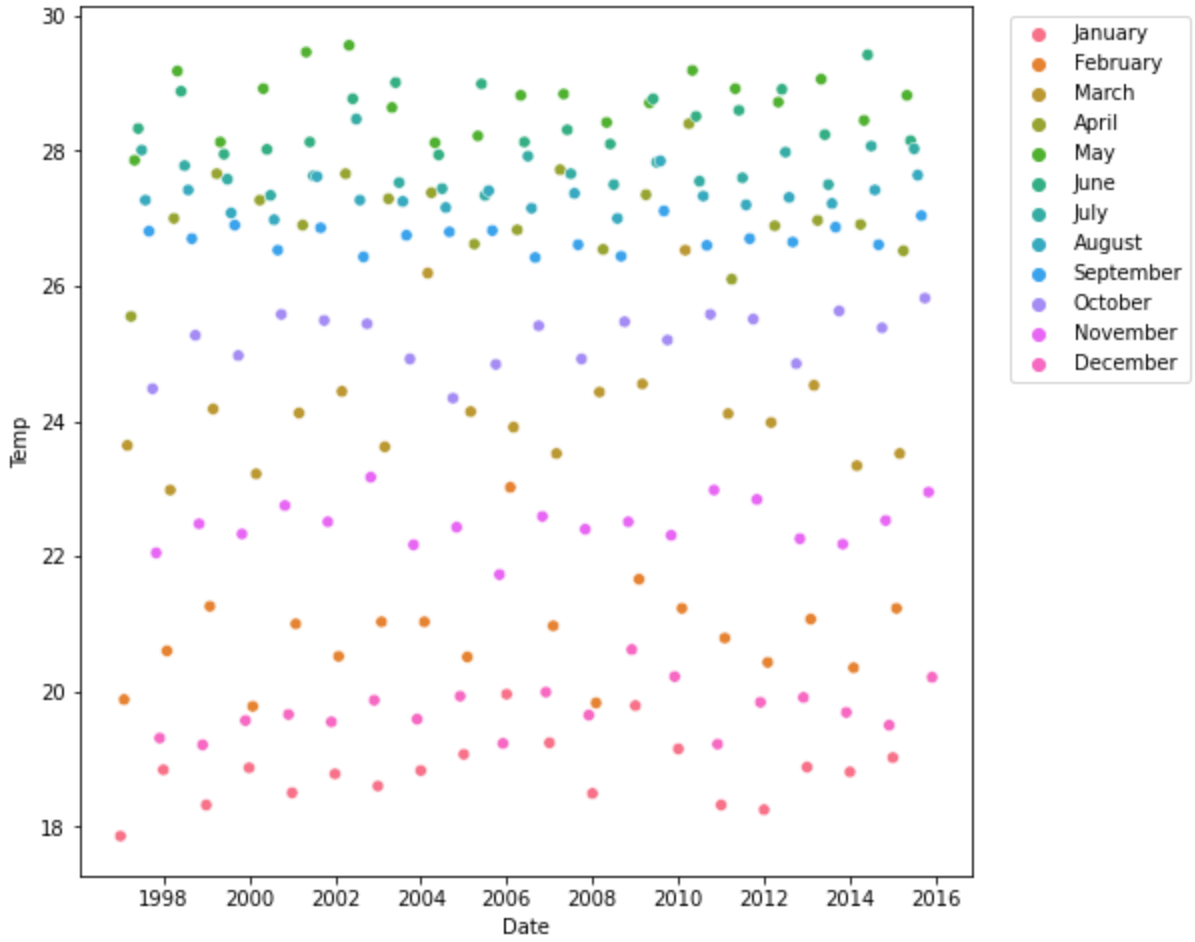
This graph mainly says that during some perids the weather value is really low and then some point in time(during that year) weather value begins to rise

Graph showing over years (1997-2015) what was the temperature per month

After checking to see if the dataset is seasonal, I decided to visualize further what was the temperature per month for each year, this gives me a good idea/summary of what the weather was like during the periods 1997-2015

In [14]:

```
fig, ax = plt.subplots(figsize = (8,8))
plot = sns.scatterplot(data = weather_df, x= weather_df['Date'], y= weather_df['Temp'],  h
plot.legend(bbox_to_anchor= (1.03, 1) );
plt.show()
```



Conclusion:

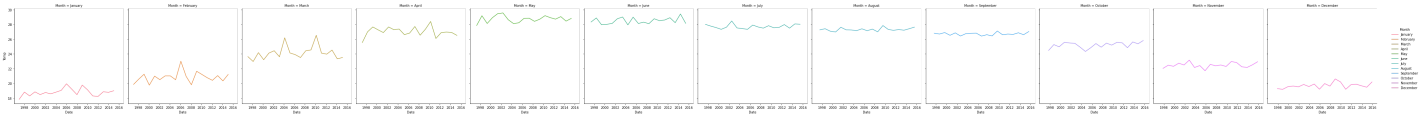
Form the graph above I can see the weather was relatively colder during the periods selected.

To better visualize this I decided I would plot this graph again but based on the crpping seasons of idnia (Summer, Autumn and Winter)

In [15]:
```
all_months = weather_df['Month'].unique()
all_months
```

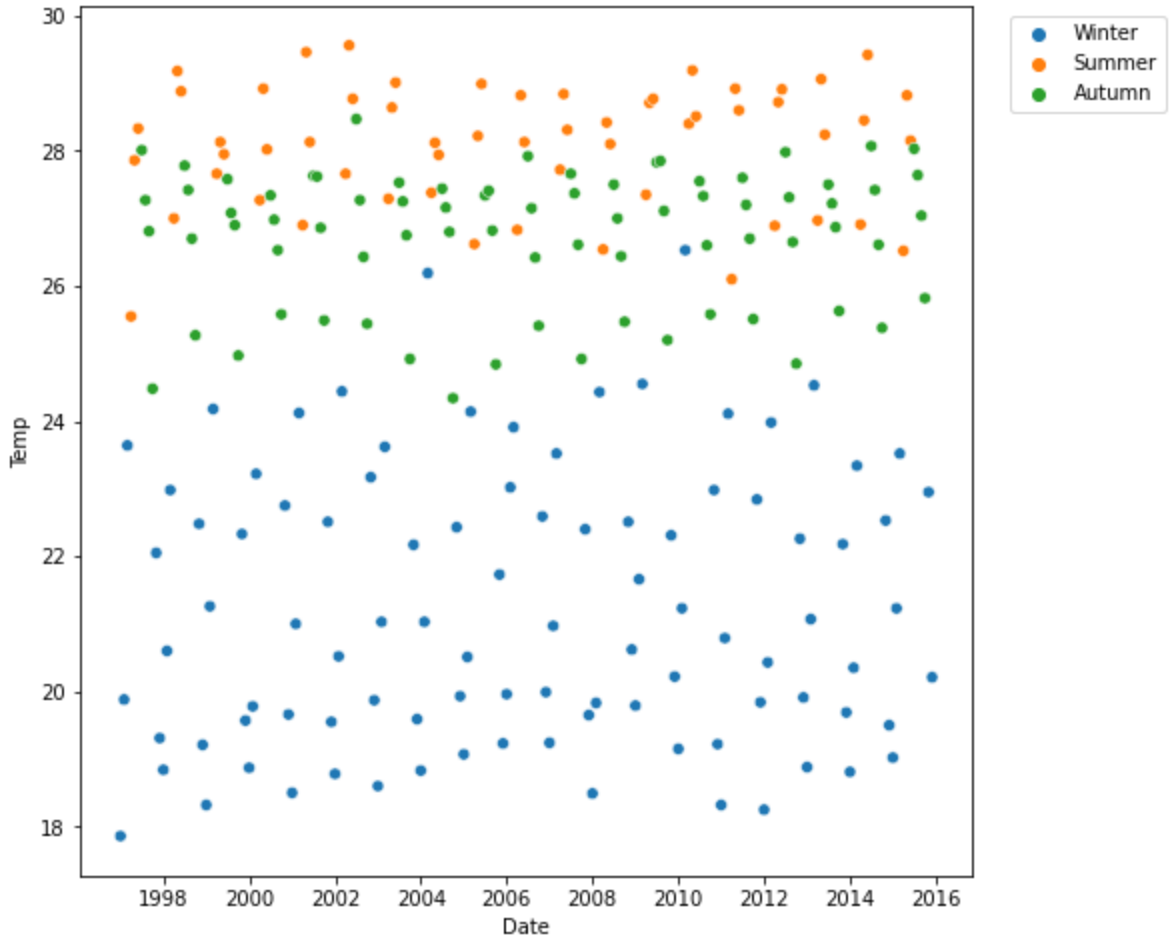
Out[15]:
```
array(['January', 'February', 'March', 'April', 'May', 'June', 'July',
       'August', 'September', 'October', 'November', 'December'],
      dtype=object)
```

In [16]:
```
plot = sns.relplot(data = weather_df, x= 'Date', y= 'Temp', col = 'Month', hue = 'Month',
plt.show()
```



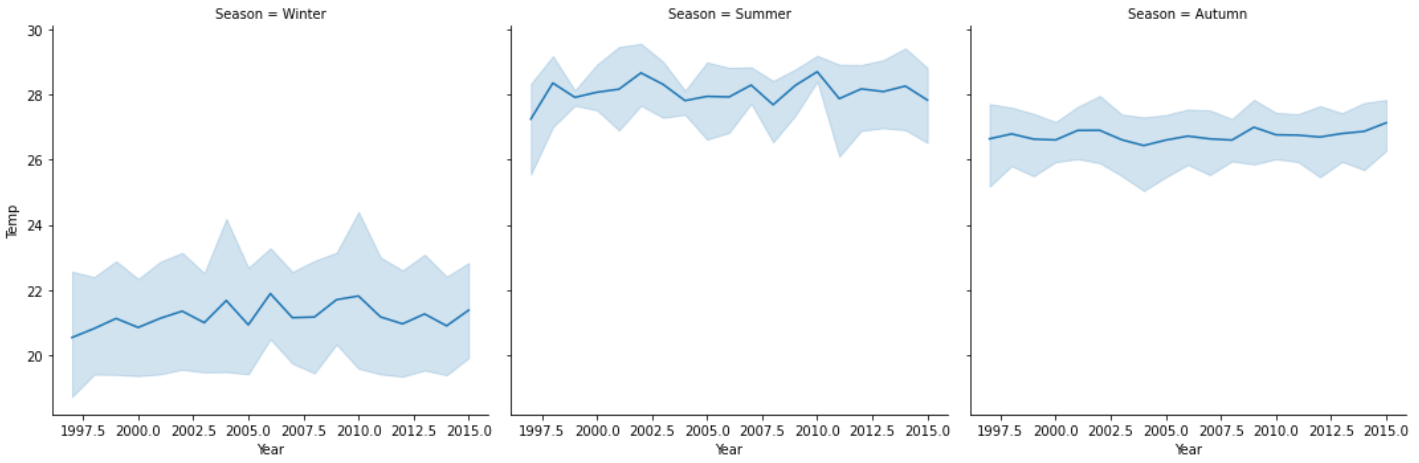
Graph showing over years (1997-2015) what was the temperature per season

In [17]:
```
fig, ax = plt.subplots(figsize = (8,8))
plot = sns.scatterplot(data = weather_df, x= 'Date', y= 'Temp',  hue='Season')
plot.legend(bbox_to_anchor= (1.03, 1) );
plt.show()
```



From the graph above we can see my inital statement above of the weather being colder in india during the periods selected is true.
We can see more points for winter(colder period)

In [18]:
```
sns.relplot(data = weather_df, x= 'Year', y= 'Temp', legend= False, col = 'Season', kind=
plt.show()
```



Line plot representation of the data side by side comparsion

Exporting the weather_df dataset

My reason for doing this is: I want to merge this dataset with the crop_production dataset in google cloud to enhance my crop_production dataset

In [19]:
```
avg_temp_per_month = weather_df.groupby(['Season', 'Year', 'Date'], as_index = False)['Ter
avg_temp_per_month
```

Out[19]:

	Season	Year	Date	Temp
0	Autumn	1997	1997-07-01	28.01
1	Autumn	1997	1997-08-01	27.27
2	Autumn	1997	1997-09-01	26.81
3	Autumn	1997	1997-10-01	24.48
4	Autumn	1998	1998-07-01	27.78
...	...	...	...	...
223	Winter	2015	2015-01-01	19.02
224	Winter	2015	2015-02-01	21.23
225	Winter	2015	2015-03-01	23.52
226	Winter	2015	2015-11-01	22.95
227	Winter	2015	2015-12-01	20.21

228 rows × 4 columns

Being that I would be working with the seasons, I decided to find the average temperature per season for each year

In [20]:

```
avg_temp = weather_df.groupby(['Season', 'Year'], as_index = False)['Temp'].mean()
avg_temp
```

Out[20]:

	Season	Year	Temp
0	Autumn	1997	26.642500
1	Autumn	1998	26.792500
2	Autumn	1999	26.632500
3	Autumn	2000	26.607500
4	Autumn	2001	26.900000
5	Autumn	2002	26.902500
6	Autumn	2003	26.612500
7	Autumn	2004	26.435000
8	Autumn	2005	26.602500
9	Autumn	2006	26.725000
10	Autumn	2007	26.640000
11	Autumn	2008	26.602500
12	Autumn	2009	26.997500
13	Autumn	2010	26.765000
14	Autumn	2011	26.752500
15	Autumn	2012	26.697500
16	Autumn	2013	26.805000

	Season	Year	Temp
17	Autumn	2014	26.870000
18	Autumn	2015	27.132500
19	Summer	1997	27.246667
20	Summer	1998	28.353333
21	Summer	1999	27.913333
22	Summer	2000	28.070000
23	Summer	2001	28.163333
24	Summer	2002	28.663333
25	Summer	2003	28.313333
26	Summer	2004	27.813333
27	Summer	2005	27.943333
28	Summer	2006	27.926667
29	Summer	2007	28.290000
30	Summer	2008	27.686667
31	Summer	2009	28.276667
32	Summer	2010	28.700000
33	Summer	2011	27.873333
34	Summer	2012	28.173333
35	Summer	2013	28.090000
36	Summer	2014	28.260000
37	Summer	2015	27.830000
38	Winter	1997	20.548000
39	Winter	1998	20.822000
40	Winter	1999	21.132000
41	Winter	2000	20.856000
42	Winter	2001	21.136000
43	Winter	2002	21.356000
44	Winter	2003	21.002000
45	Winter	2004	21.682000
46	Winter	2005	20.936000
47	Winter	2006	21.894000
48	Winter	2007	21.156000
49	Winter	2008	21.176000
50	Winter	2009	21.706000
51	Winter	2010	21.822000
52	Winter	2011	21.180000

	Season	Year	Temp
53	Winter	2012	20.966000
54	Winter	2013	21.270000
55	Winter	2014	20.906000
56	Winter	2015	21.386000

In [21]:
```
avg_temp.to_csv('india_weather_data_1997_2015.csv', index = True)
```

Modelling(Working with time series)

I realised after more analyses on my main dataset that I would have to use time seies approach to attain my final result, therefore I decided to work with time series algorthim (ARIMA) with this dataset being that i found the dataset is seasonal

Important Note

When looking to fit time series data with a seasonal ARIMA model, the first goal is to find the values of ARIMA(p,d,q)(P,D,Q)s

Selecting the optimal parameter values for our ARIMA(p,d,q)(P,D,Q)time series model

Step 1:

This will be done by using grid search to iteratively explore different combinations of parametes.

Step 2:

We will then fit a new Seasonal ARIMA model with the SARIMAX() function from statsmodel module and assess its overall quality

Step 3:

After exploring entire landscape parameters, the optimal set of parameters will be the one that yields the best performance for our criteria of interest

Beginning

Step 1:

Generating the various combination of parameters that are needed to be assessed

In [22]:
```
test_df = weather_df.drop('Year', axis = 'columns')
test_df.drop('Month', axis = 'columns', inplace=True)
test_df.drop('Season', axis = 'columns', inplace = True)
```

In [23]:
```
test_df = test_df.set_index('Date')
```

In [24]:
```
import warnings
import itertools
import statsmodels.api as sm
```

In [25]:
```
# Define the p, d and q parameters to take any value between 0 and 2
p = d = q = range(0, 2)

# Generate all different combinations of p, d and q triplets
pdq = list(itertools.product(p, d, q))

# Generate all different combinations of seasonal p, q and q triplets
seasonal_pdq = [(x[0], x[1], x[2], 12) for x in list(itertools.product(p, d, q))]

print('Examples of parameter combinations for Seasonal ARIMA...')
print('SARIMAX: {} x {}'.format(pdq[1], seasonal_pdq[1]))
print('SARIMAX: {} x {}'.format(pdq[1], seasonal_pdq[2]))
print('SARIMAX: {} x {}'.format(pdq[2], seasonal_pdq[3]))
print('SARIMAX: {} x {}'.format(pdq[2], seasonal_pdq[4]))
```

```
Examples of parameter combinations for Seasonal ARIMA...
SARIMAX: (0, 0, 1) x (0, 0, 1, 12)
SARIMAX: (0, 0, 1) x (0, 1, 0, 12)
SARIMAX: (0, 1, 0) x (0, 1, 1, 12)
SARIMAX: (0, 1, 0) x (1, 0, 0, 12)
```

Training and Evaluating ARIMA models on different combinations

The AIC (Akaike Information Criterion) value is used to measure how well a model fits the data while taking into account the overall complexity of the model

Models which fit the data very well while using lots of features will be assigned a larger AIC score than a model that uses fewer features to acheive the same goodness-of-fit

Best practise:

Look for models that yield the lowest AIC value

Step 2:

Assessing each parameter generated by the use of SARIMAX fucntion from statsmodels to fit the corresponding Seasonal ARIMA model.

After fitting each SARIMAX() model, the code prints out its respective AIC score

In [26]:
```
warnings.filterwarnings("ignore") # specify to ignore warning messages

# the values of pdq are generated and stored above step (Step1)
for param in pdq:
    # the values of seasonal_pdq are generated and stored above step (Step1)
    for param_seasonal in seasonal_pdq:
        try:
            # initializing the SARIMAX model and assigning its pdq parameters and seasonal
            # which have been generated with itertools
            mod = sm.tsa.statespace.SARIMAX(test_df,
                                            order=param,
                                            seasonal_order=param_seasonal,
                                            enforce_stationarity=False,
                                            enforce_invertibility=False)

            # storing the AIC values for each param(pdq and seasonal pdq)
            results = mod.fit()
            print('SARIMAX{}x{}12 - AIC:{}'.format(param, param_seasonal, results.aic))
        except:
            continue
```

```
SARIMAX(0, 0, 0)x(0, 0, 0, 12)12 - AIC:2108.1166673989255
SARIMAX(0, 0, 0)x(0, 0, 1, 12)12 - AIC:1748.5292899710907
SARIMAX(0, 0, 0)x(0, 1, 0, 12)12 - AIC:461.3133820034569
SARIMAX(0, 0, 0)x(0, 1, 1, 12)12 - AIC:323.4784456437806
SARIMAX(0, 0, 0)x(1, 0, 0, 12)12 - AIC:466.1386478622528
SARIMAX(0, 0, 0)x(1, 0, 1, 12)12 - AIC:346.41578499185346
SARIMAX(0, 0, 0)x(1, 1, 0, 12)12 - AIC:379.92851562205806
SARIMAX(0, 0, 0)x(1, 1, 1, 12)12 - AIC:321.371618461159
SARIMAX(0, 0, 1)x(0, 0, 0, 12)12 - AIC:1795.3725895739155
SARIMAX(0, 0, 1)x(0, 0, 1, 12)12 - AIC:1460.9155183448206
SARIMAX(0, 0, 1)x(0, 1, 0, 12)12 - AIC:442.05700388514504
SARIMAX(0, 0, 1)x(0, 1, 1, 12)12 - AIC:295.5423179905438
SARIMAX(0, 0, 1)x(1, 0, 0, 12)12 - AIC:469.8946742971925
SARIMAX(0, 0, 1)x(1, 0, 1, 12)12 - AIC:320.79102728580784
SARIMAX(0, 0, 1)x(1, 1, 0, 12)12 - AIC:359.978551962295
SARIMAX(0, 0, 1)x(1, 1, 1, 12)12 - AIC:297.0954431573401
SARIMAX(0, 1, 0)x(0, 0, 0, 12)12 - AIC:962.4425266685311
SARIMAX(0, 1, 0)x(0, 0, 1, 12)12 - AIC:767.6899459329384
SARIMAX(0, 1, 0)x(0, 1, 0, 12)12 - AIC:537.8224169243906
SARIMAX(0, 1, 0)x(0, 1, 1, 12)12 - AIC:372.2856284616105
SARIMAX(0, 1, 0)x(1, 0, 0, 12)12 - AIC:531.363810870643
SARIMAX(0, 1, 0)x(1, 0, 1, 12)12 - AIC:406.6604190449842
SARIMAX(0, 1, 0)x(1, 1, 0, 12)12 - AIC:442.57802550699216
SARIMAX(0, 1, 0)x(1, 1, 1, 12)12 - AIC:381.2003313118639
SARIMAX(0, 1, 1)x(0, 0, 0, 12)12 - AIC:820.622346777964
SARIMAX(0, 1, 1)x(0, 0, 1, 12)12 - AIC:715.5160765218941
SARIMAX(0, 1, 1)x(0, 1, 0, 12)12 - AIC:463.5375935139289
SARIMAX(0, 1, 1)x(0, 1, 1, 12)12 - AIC:319.8210302445243
SARIMAX(0, 1, 1)x(1, 0, 0, 12)12 - AIC:468.40702236845783
SARIMAX(0, 1, 1)x(1, 0, 1, 12)12 - AIC:344.8483889720818
SARIMAX(0, 1, 1)x(1, 1, 0, 12)12 - AIC:386.0101597141644
SARIMAX(0, 1, 1)x(1, 1, 1, 12)12 - AIC:321.71832126390404
SARIMAX(1, 0, 0)x(0, 0, 0, 12)12 - AIC:968.3824908405162
SARIMAX(1, 0, 0)x(0, 0, 1, 12)12 - AIC:773.2246337172405
SARIMAX(1, 0, 0)x(0, 1, 0, 12)12 - AIC:445.4294454838282
SARIMAX(1, 0, 0)x(0, 1, 1, 12)12 - AIC:298.00474458676786
SARIMAX(1, 0, 0)x(1, 0, 0, 12)12 - AIC:447.39230066321977
SARIMAX(1, 0, 0)x(1, 0, 1, 12)12 - AIC:322.8543563206463
SARIMAX(1, 0, 0)x(1, 1, 0, 12)12 - AIC:360.0017400828415
SARIMAX(1, 0, 0)x(1, 1, 1, 12)12 - AIC:301.9641461263683
SARIMAX(1, 0, 1)x(0, 0, 0, 12)12 - AIC:829.4440728358886
SARIMAX(1, 0, 1)x(0, 0, 1, 12)12 - AIC:721.9180656056305
SARIMAX(1, 0, 1)x(0, 1, 0, 12)12 - AIC:444.01048896073917
SARIMAX(1, 0, 1)x(0, 1, 1, 12)12 - AIC:296.04546397449946
SARIMAX(1, 0, 1)x(1, 0, 0, 12)12 - AIC:448.13357835656393
SARIMAX(1, 0, 1)x(1, 0, 1, 12)12 - AIC:321.0981400591917
SARIMAX(1, 0, 1)x(1, 1, 0, 12)12 - AIC:361.02071856283567
SARIMAX(1, 0, 1)x(1, 1, 1, 12)12 - AIC:299.0774306650603
SARIMAX(1, 1, 0)x(0, 0, 0, 12)12 - AIC:819.7393282112212
SARIMAX(1, 1, 0)x(0, 0, 1, 12)12 - AIC:707.3214122666176
SARIMAX(1, 1, 0)x(0, 1, 0, 12)12 - AIC:521.218776037205
SARIMAX(1, 1, 0)x(0, 1, 1, 12)12 - AIC:358.4229612152076
SARIMAX(1, 1, 0)x(1, 0, 0, 12)12 - AIC:516.787300492738
SARIMAX(1, 1, 0)x(1, 0, 1, 12)12 - AIC:391.45559970428974
SARIMAX(1, 1, 0)x(1, 1, 0, 12)12 - AIC:421.1534892105016
SARIMAX(1, 1, 0)x(1, 1, 1, 12)12 - AIC:366.65468500143703
SARIMAX(1, 1, 1)x(0, 0, 0, 12)12 - AIC:782.6482846510519
SARIMAX(1, 1, 1)x(0, 0, 1, 12)12 - AIC:701.4249044266761
SARIMAX(1, 1, 1)x(0, 1, 0, 12)12 - AIC:447.32542984384725
SARIMAX(1, 1, 1)x(0, 1, 1, 12)12 - AIC:301.57076841349897
SARIMAX(1, 1, 1)x(1, 0, 0, 12)12 - AIC:449.4029514406069
SARIMAX(1, 1, 1)x(1, 0, 1, 12)12 - AIC:328.5628801545914
SARIMAX(1, 1, 1)x(1, 1, 0, 12)12 - AIC:363.11234826541244
SARIMAX(1, 1, 1)x(1, 1, 1, 12)12 - AIC:305.72310772687456
```

The output after assessing each parameter gives that the best parameter for predicting is

SARIMAX(0, 0, 1)x(0, 1, 1, 12)

its AIC: 295.5423179905438 which is the lowest

Fitting an ARIMA Time Series Model

In [27]:
```
# model initialization and fitting
mod = sm.tsa.statespace.SARIMAX(test_df,
                                order = (0, 0, 1),
                                seasonal_order = (0, 1, 1, 12),
                                enforce_stationarity=False,
                                enforce_invertibility=False)

results = mod.fit()
results.summary().tables[1]
```

Out[27]:

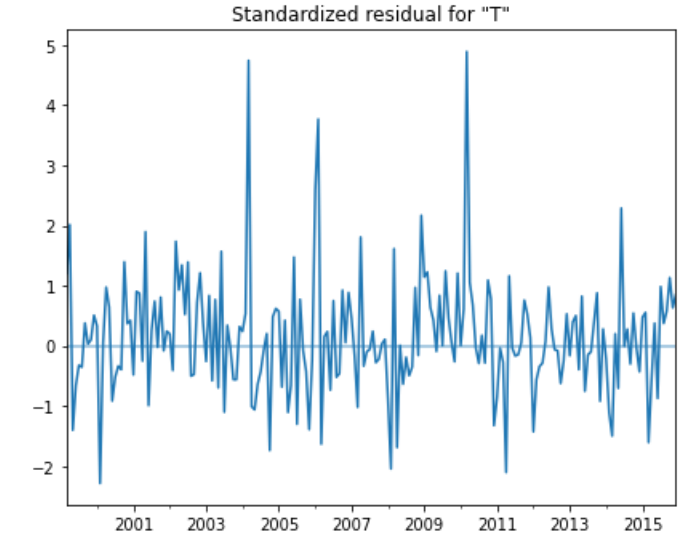
	coef	std err	z	P>|z|	[0.025	0.975]
ma.L1	0.3541	0.062	5.744	0.000	0.233	0.475
ma.S.L12	-1.0000	232.655	-0.004	0.997	-456.995	454.995
sigma2	0.2148	49.993	0.004	0.997	-97.769	98.198

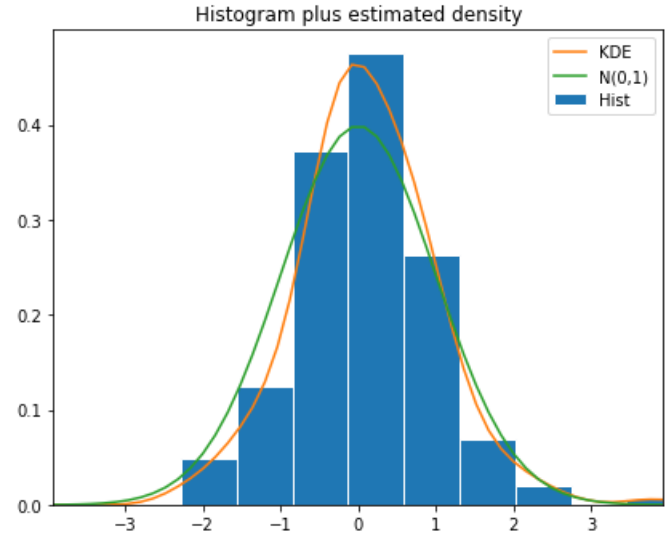
Important

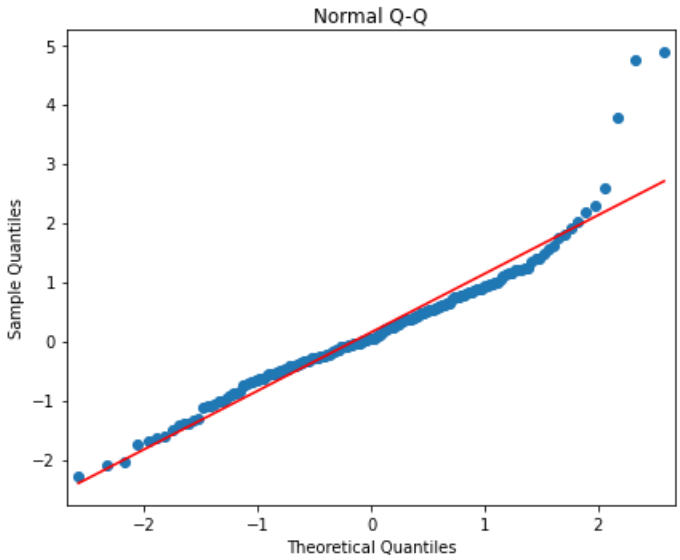
After fitting models it is important to run model diagnostics to ensure that none of the assumptions made by the model have been violated

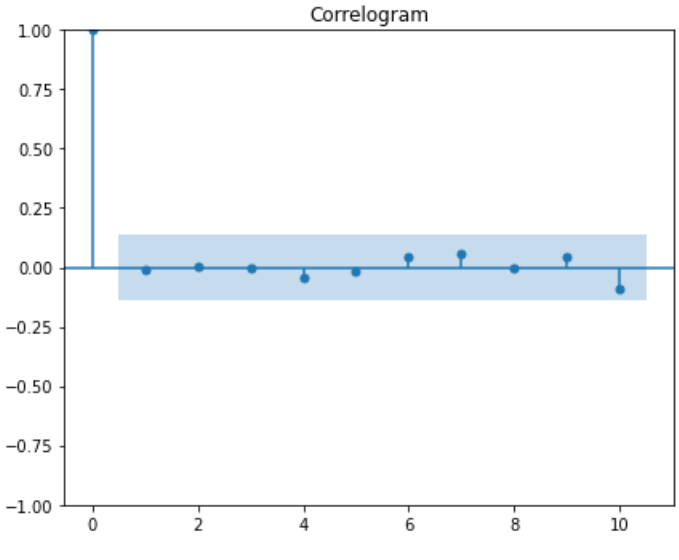
Model Diagnostics

In [28]:
```
results.plot_diagnostics(figsize=(15, 12))
plt.show()
```









In [29]:
```
test_df
```

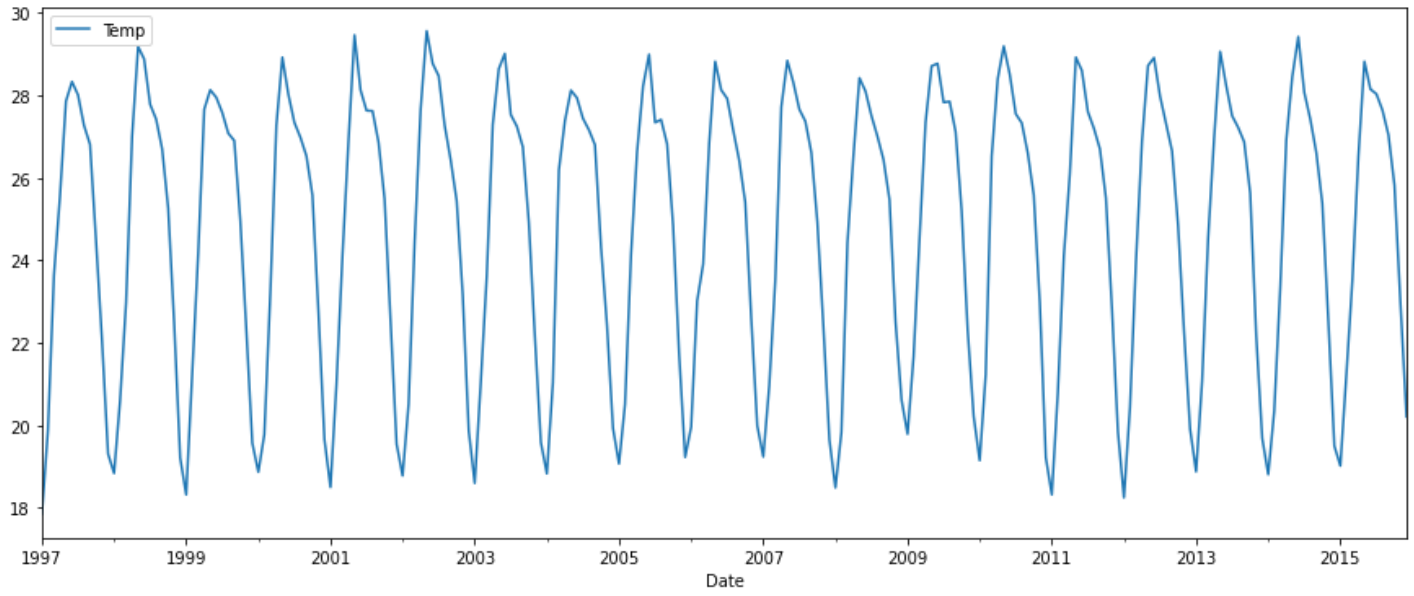
Out[29]:

Date	Temp
1997-01-01	17.86
1997-02-01	19.88
1997-03-01	23.64
1997-04-01	25.55
1997-05-01	27.86
...	...
2015-08-01	27.64
2015-09-01	27.04
2015-10-01	25.82
2015-11-01	22.95
2015-12-01	20.21

228 rows × 1 columns

Validating Forecasts

In [30]:
```
test_df.plot(figsize=(15, 6))
plt.show()
```



In [31]:
```
test_df['2001':]
```

Out[31]:

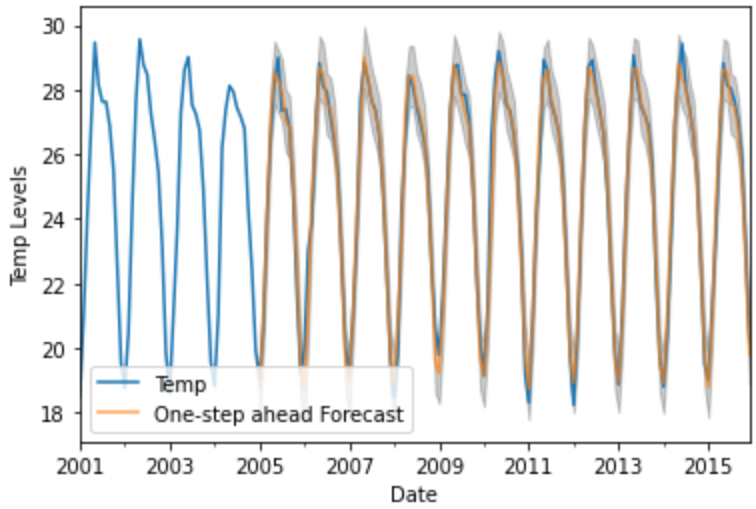
Date	Temp
2001-01-01	18.50
2001-02-01	21.00
2001-03-01	24.12
2001-04-01	26.90
2001-05-01	29.46
...	...
2015-08-01	27.64
2015-09-01	27.04
2015-10-01	25.82
2015-11-01	22.95
2015-12-01	20.21

180 rows × 1 columns

In [32]:
```
pred = results.get_prediction(start=pd.to_datetime('2005-01-01'), dynamic=False)
pred_ci = pred.conf_int()
```

In [33]:
```
ax = test_df['2001':].plot(label='observed')
pred.predicted_mean.plot(ax=ax, label='One-step ahead Forecast', alpha=.7)

ax.fill_between(pred_ci.index,
                pred_ci.iloc[:, 0],
```

```
                pred_ci.iloc[:, 1], color='k', alpha=.2)

ax.set_xlabel('Date')
ax.set_ylabel('Temp Levels')
plt.legend()

plt.show()
```



Above we can see that my model forecasting after using the parameters recommended by grid search shows its forecast are not too far-fetched from the test data forecast therefore this model can be used to make future prediction

Conclusion

After the results above, I believe I have a good idea on how to work with arima for forecasting data