Predict Temperature Of Different Cities Using Time Series Algorithmns

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

In [2]:
```
global_temp = pd.read_csv(r'C:\Users\hp\Downloads\drive-download-20230417T062356Z-001\G
```

In [3]:
```
global_temp.head(3)
```

Out[3]:

	dt	LandAverageTemperature	LandAverageTemperatureUncertainty	LandMaxTemperature	LandMax
0	1750-01-01	3.034	3.574	NaN	
1	1750-02-01	3.083	3.702	NaN	
2	1750-03-01	5.626	3.076	NaN	

In [4]:
```
global_temp.duplicated().sum()
```

Out[4]:
```
0
```

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

Out[5]:
```
dt                                           object
LandAverageTemperature                      float64
LandAverageTemperatureUncertainty           float64
LandMaxTemperature                          float64
LandMaxTemperatureUncertainty               float64
LandMinTemperature                          float64
LandMinTemperatureUncertainty               float64
LandAndOceanAverageTemperature              float64
LandAndOceanAverageTemperatureUncertainty   float64
dtype: object
```

In [6]:
```
global_temp['dt'][0]
```

Out[6]:
```
'1750-01-01'
```

In [7]:
```
global_temp['dt']= pd.to_datetime(global_temp['dt'])
```

In [8]:
```
global_temp['dt'].dtype
```

Out[8]:
```
dtype('<M8[ns]')
```

Analysing Whether Global Warming Exists Or Not

In [9]:
```
global_temp['dt'].dt.year
```

Out[9]:
```
0       1750
1       1750
2       1750
3       1750
4       1750
        ...
3187    2015
3188    2015
3189    2015
3190    2015
3191    2015
Name: dt, Length: 3192, dtype: int64
```

In [10]:
```
global_temp['years']=global_temp['dt'].dt.year
```

In [11]:
```
global_temp.head(2)
```

Out[11]:

	dt	LandAverageTemperature	LandAverageTemperatureUncertainty	LandMaxTemperature	LandMax
0	1750-01-01	3.034	3.574	NaN	
1	1750-02-01	3.083	3.702	NaN	

In [12]:
```
data = global_temp.groupby('years').agg({'LandAverageTemperature':'mean','LandAverageTe
```

In [13]:
```
data.columns
```

Out[13]:
```
Index(['years', 'LandAverageTemperature', 'LandAverageTemperatureUncertainty'], dtype='o
bject')
```

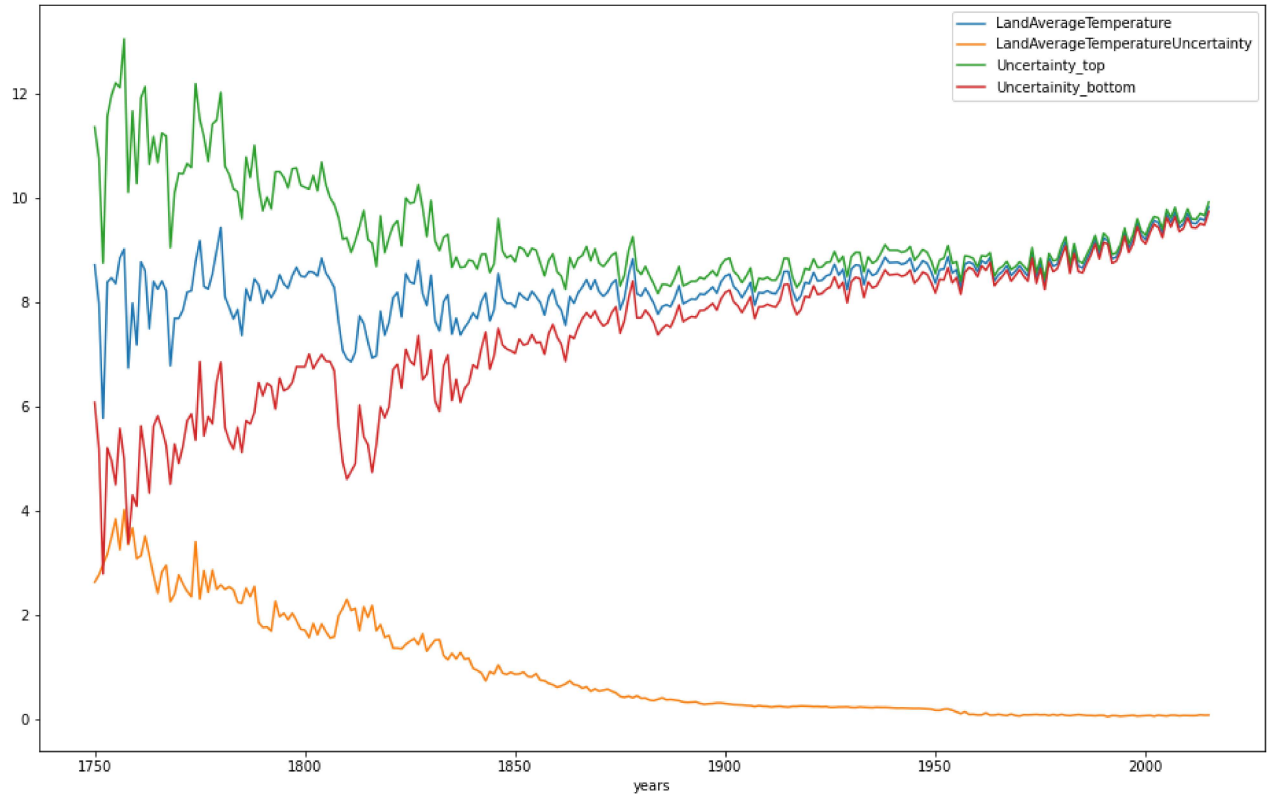
In [14]:
```
data['Uncertainty_top']=data['LandAverageTemperature'] + data['LandAverageTemperatureUn
data['Uncertainity_bottom']=data['LandAverageTemperature'] - data['LandAverageTemperatu
```

In [15]:
```
data.columns
```

Out[15]:
```
Index(['years', 'LandAverageTemperature', 'LandAverageTemperatureUncertainty',
       'Uncertainty_top', 'Uncertainity_bottom'],
      dtype='object')
```

In [16]:
```
data.plot(x='years',y=['LandAverageTemperature','LandAverageTemperatureUncertainty',
       'Uncertainty_top','Uncertainity_bottom'],figsize=(16,10))
```

Out[16]:
```
<AxesSubplot:xlabel='years'>
```



EXPLORE TEMPERATURE FOR DIFFERENT SEASONS

In [17]:
```
global_temp['month']=global_temp['dt'].dt.month
```

In [18]:
```
global_temp['month'].unique()
```

Out[18]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12], dtype=int64)
```

In [19]:
```
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'
```

In [20]:
```
global_temp['month'].apply(get_season)
```

Out[20]:
```
0       winter
1       winter
2       spring
3       spring
4       spring
         ...
3187    summer
3188    autumn
3189    autumn
```

```
3190    autumn
3191    winter
Name: month, Length: 3192, dtype: object
```

In [21]:
```
global_temp['season']=global_temp['month'].apply(get_season)
```

In [22]:
```
global_temp['season']
```

Out[22]:
```
0       winter
1       winter
2       spring
3       spring
4       spring
         ...
3187    summer
3188    autumn
3189    autumn
3190    autumn
3191    winter
Name: season, Length: 3192, dtype: object
```

In [23]:
```
global_temp['season'].unique()
```

Out[23]:
```
array(['winter', 'spring', 'summer', 'autumn'], dtype=object)
```

In [24]:
```
years=global_temp['years'].unique()
```

In [25]:
```
spring_temp=[]
summer_temp=[]
autumn_temp=[]
winter_temp=[]

for year in years:
    current_yr=global_temp[global_temp['years']==year]


    spring_temp.append(current_yr[current_yr['season']=='spring']['LandAverageTemperatu
    summer_temp.append(current_yr[current_yr['season']=='summer']['LandAverageTemperatu
    autumn_temp.append(current_yr[current_yr['season']=='autumn']['LandAverageTemperatu
    winter_temp.append(current_yr[current_yr['season']=='winter']['LandAverageTemperatu
```

In [26]:
```
season=pd.DataFrame()
```

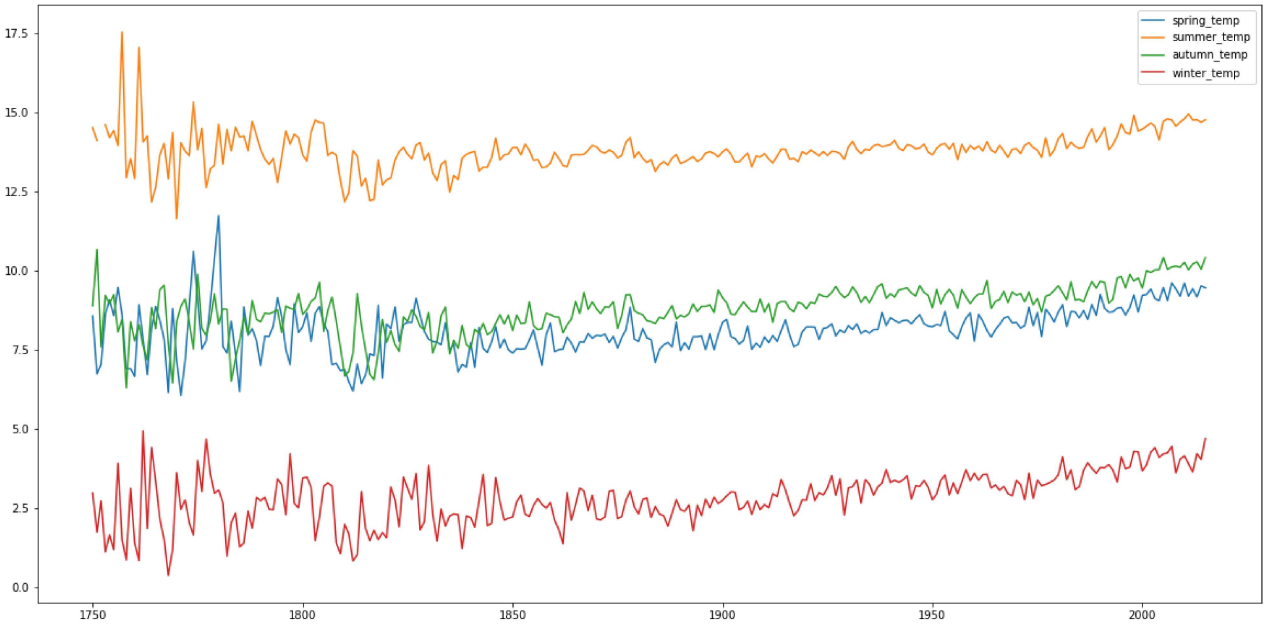
In [27]:
```
season['years']=years
season['spring_temp']=spring_temp
season['summer_temp']=summer_temp
season['autumn_temp']=autumn_temp
season['winter_temp']=winter_temp
```

In [28]:
```
season.columns
```

Out[28]:
```
Index(['years', 'spring_temp', 'summer_temp', 'autumn_temp', 'winter_temp'], dtype='obje
ct')
```

In [29]:
```
plt.figure(figsize=(20,10))
plt.plot(season['years'],season[['spring_temp', 'summer_temp', 'autumn_temp', 'winter_t
plt.legend(['spring_temp','summer_temp','autumn_temp','winter_temp'])
```

Out[29]: <matplotlib.legend.Legend at 0xc20f208>



In [30]:
```
cities=pd.read_csv(r'C:\Users\hp\Downloads\drive-download-20230417T062356Z-001\GlobalLa
```

In [31]:
```
cities.shape
```

Out[31]: (8599212, 7)

In [32]:
```
cities.columns
```

Out[32]:
```
Index(['dt', 'AverageTemperature', 'AverageTemperatureUncertainty', 'City',
       'Country', 'Latitude', 'Longitude'],
      dtype='object')
```

In [33]:
```
usa=cities[cities['Country']=='United States']
```

In [34]:
```
usa.columns
```

Out[34]:
```
Index(['dt', 'AverageTemperature', 'AverageTemperatureUncertainty', 'City',
       'Country', 'Latitude', 'Longitude'],
      dtype='object')
```

In [35]:
```
usa_cities=['New York','Los Angeles','San Francisco']
```

In [36]:
```
data2=usa[usa['City'].isin(usa_cities)]
```

In [37]:
```
data2.columns
```

Out[37]:
```
Index(['dt', 'AverageTemperature', 'AverageTemperatureUncertainty', 'City',
       'Country', 'Latitude', 'Longitude'],
      dtype='object')
```

In [38]:
```
data2=data2[['dt','AverageTemperature']]
```

In [39]:
```
data2.columns=['Date','Temp']
```

In [40]:
```
data2.head(3)
```

Out[40]:

	Date	Temp
4356748	1849-01-01	8.819
4356749	1849-02-01	9.577
4356750	1849-03-01	11.814

In [41]:
```
data2.dtypes
```

Out[41]:
```
Date     object
Temp    float64
dtype: object
```

In [42]:
```
data2['Date']=pd.to_datetime(data2['Date'])
```

In [43]:
```
data2.isnull().sum()
```

Out[43]:
```
Date      0
Temp    120
dtype: int64
```

In [44]:
```
data2.dropna(inplace=True)
```

In [45]:
```
data2.head(3)
```

Out[45]:

	Date	Temp
4356748	1849-01-01	8.819
4356749	1849-02-01	9.577
4356750	1849-03-01	11.814

In [46]:
```
data2.set_index('Date',inplace=True)
```

In [47]:
```
data2.head(3)
```

Out[47]:

	Temp
Date	
1849-01-01	8.819
1849-02-01	9.577
1849-03-01	11.814

Check whether data has seasonality or not

In [48]:
```
data2['year']=data2.index.year
```

In [49]:
```
data2['month']=data2.index.month
```

In [50]:
```
pivot=data2.pivot_table(values='Temp',index='month',columns='year')
```

In [51]:
```
pivot
```

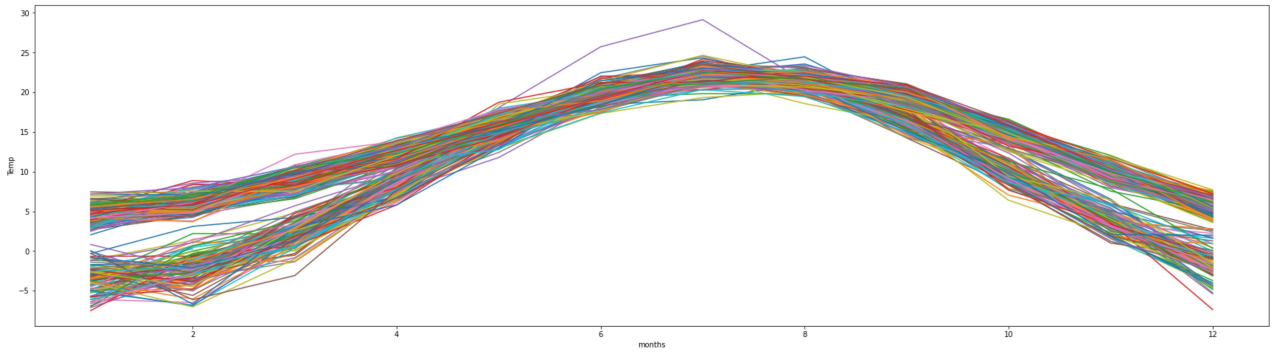
Out[51]:

year	1743	1744	1745	1750	1751	1752	1753	1754	1755	1756	...	2004	2
month													
1	NaN	NaN	-2.363	-4.310	-3.591	-7.588	-3.122	-2.252	-3.193	-1.985	...	4.080333	4.92
2	NaN	NaN	-2.671	-2.719	-2.051	NaN	-1.467	-2.583	-1.802	0.259	...	6.100000	7.29
3	NaN	NaN	1.363	2.773	3.256	3.322	4.207	2.728	1.112	NaN	...	12.184000	9.12
4	NaN	9.788	8.209	8.848	7.992	7.402	8.099	NaN	8.714	NaN	...	13.806333	12.54
5	NaN	15.708	NaN	15.411	NaN	NaN	15.330	NaN	15.238	NaN	...	17.817333	15.98
6	NaN	21.210	NaN	19.017	20.724	NaN	20.820	20.075	19.964	20.488	...	19.872000	19.77
7	NaN	22.207	NaN	24.203	22.668	NaN	22.524	22.503	NaN	22.452	...	22.246333	23.61
8	NaN	NaN	NaN	22.135	21.547	NaN	21.324	21.461	NaN	21.208	...	22.101333	23.02
9	NaN	14.922	NaN	17.445	15.812	NaN	15.548	16.281	16.137	17.345	...	20.669333	19.42
10	NaN	8.968	NaN	9.076	NaN	9.391	10.479	11.477	8.669	9.662	...	14.205667	15.09
11	3.264	3.161	NaN	NaN	NaN	5.831	3.363	NaN	3.599	2.894	...	9.758333	11.73
12	NaN	-2.681	NaN	-1.093	NaN	-1.471	-2.854	-0.752	-2.381	-2.900	...	6.428000	6.58

12 rows × 266 columns

In [52]:
```
pivot.plot(figsize=(30,8))
plt.legend().remove()
plt.xlabel('months')
plt.ylabel('Temp')
```

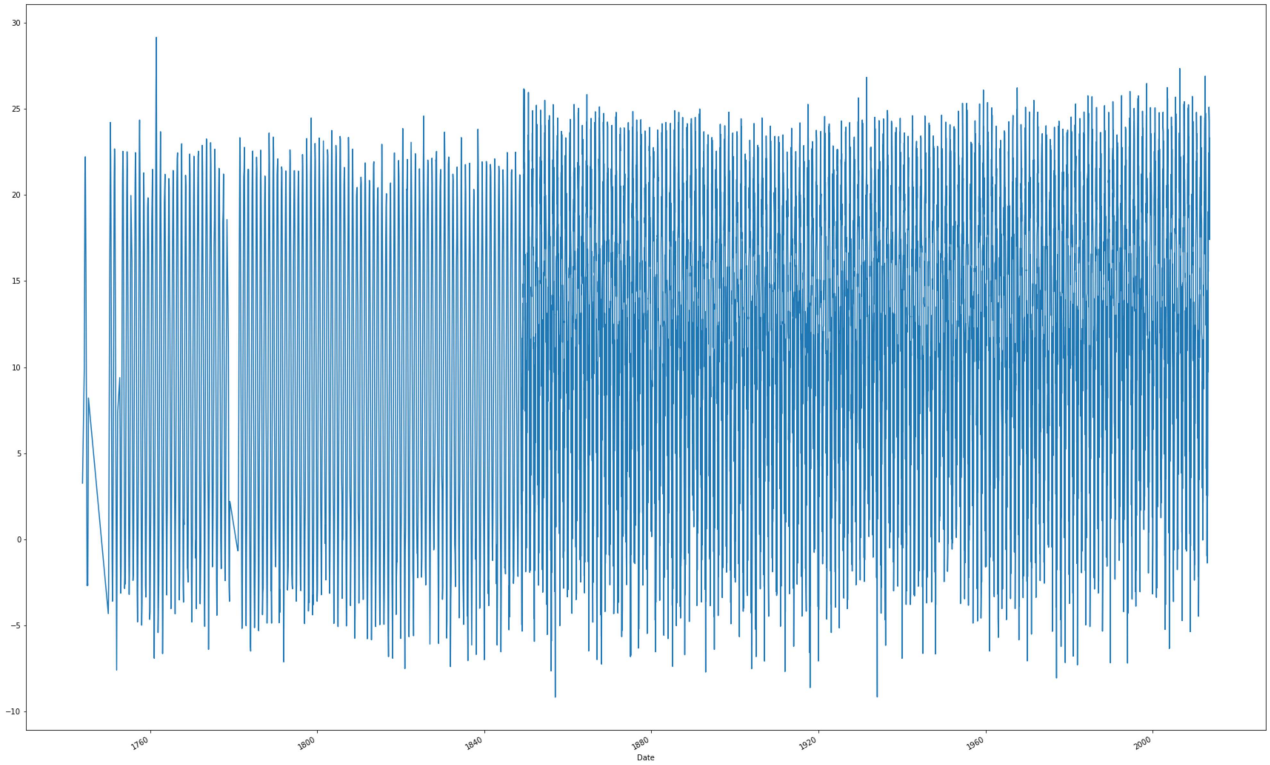
Out[52]:
```
Text(0, 0.5, 'Temp')
```



Check whether data is stationary or not

In [53]:
```
data2['Temp'].plot(figsize=(30,20))
```

Out[53]:
```
<AxesSubplot:xlabel='Date'>
```



In [54]:
```
from statsmodels.tsa.stattools import adfuller
```

In [55]:
```
adfuller(data2['Temp'])
```

Out[55]:
```
(-2.006389303675878,
 0.28377865833329,
 35,
 7037,
 {'1%': -3.431279613044704,
  '5%': -2.8619508146206374,
  '10%': -2.5669886726127307},
 28011.222669408726)
```

In [56]:
```
def adfuller_test(data):
    result=adfuller(data)
    labels=['ADF Test Stats','p-value','lags_used','nobs']
    for value, label in zip(result,labels):
     print(label+':'+str(value))
    if result[1] <= 0.05:
     print('reject the HO,data is stationary')
    else:
     print('fail to reject HO, data is not stationary')
```

In [57]:
```
adfuller_test(data2['Temp'])
```

```
ADF Test Stats:-2.006389303675878
p-value:0.28377865833329
lags_used:35
nobs:7037
fail to reject HO, data is not stationary
```

How to make the data stationary

In [58]:
```
data2['Temp'].head(13)
```

Out[58]:
```
Date
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
1849-06-01    21.173
1849-07-01    26.159
1849-08-01    26.099
1849-09-01    21.848
1849-10-01    16.549
1849-11-01    10.907
1849-12-01     7.032
1850-01-01     7.087
Name: Temp, dtype: float64
```

In [59]:
```
data2['Temp'].shift(12).head(13)
```

Out[59]:
```
Date
1849-01-01      NaN
1849-02-01      NaN
1849-03-01      NaN
1849-04-01      NaN
1849-05-01      NaN
1849-06-01      NaN
1849-07-01      NaN
1849-08-01      NaN
1849-09-01      NaN
1849-10-01      NaN
1849-11-01      NaN
1849-12-01      NaN
1850-01-01    8.819
Name: Temp, dtype: float64
```

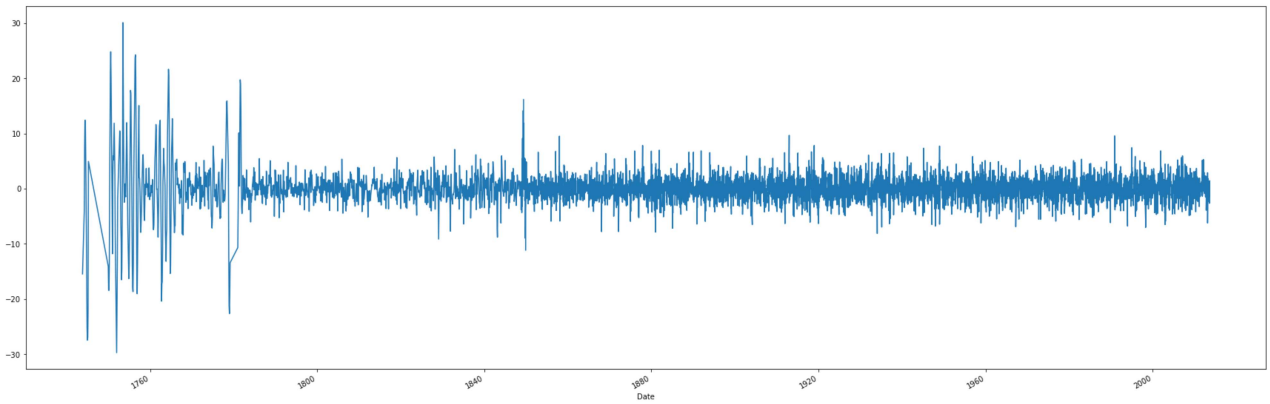
In [60]:
```
data2['first_diff_temp']=data2['Temp']-data2['Temp'].shift(12)
```

In [61]:
```
adfuller_test(data2['first_diff_temp'].dropna())
```

```
ADF Test Stats:-21.23965040491093
p-value:0.0
lags_used:35
nobs:7025
reject the HO,data is stationary
```

In [62]:
```
data2['first_diff_temp'].plot(figsize=(30,10))
```

Out[62]:
```
<AxesSubplot:xlabel='Date'>
```



build time series model using moving averages

In [63]:
```
df=data2[['first_diff_temp']]
```

In [64]:
```
import warnings
from warnings import filterwarnings
filterwarnings('ignore')
```

In [65]:
```
df.dropna(inplace=True)
```

In [66]:
```
df['Predictions']=df['first_diff_temp'].rolling(window=5).mean()
```

In [67]:
```
df.head(3)
```

Out[67]:

Date	first_diff_temp	Predictions
1850-01-01	-1.732	NaN
1850-02-01	-1.002	NaN
1850-03-01	-1.449	NaN

In [68]:
```
df.dropna(inplace=True)
```

In [69]:
```
df.head(3)
```

Out[69]:

Date	first_diff_temp	Predictions
1850-05-01	1.799	-0.4706
1850-06-01	-0.932	-0.3106
1850-07-01	-2.714	-0.6530

In [70]:
```
df.columns=['actual_temp','forcasted_temp']
```

In [71]:
```
from sklearn.metrics import mean_squared_error
```

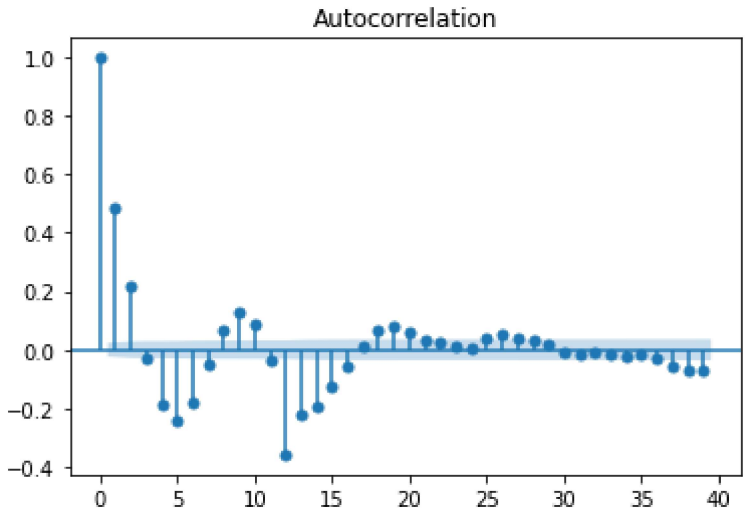
In [72]:
```
np.sqrt(mean_squared_error(df['actual_temp'],df['forcasted_temp']))
```

Out[72]:
```
2.3934235122562058
```

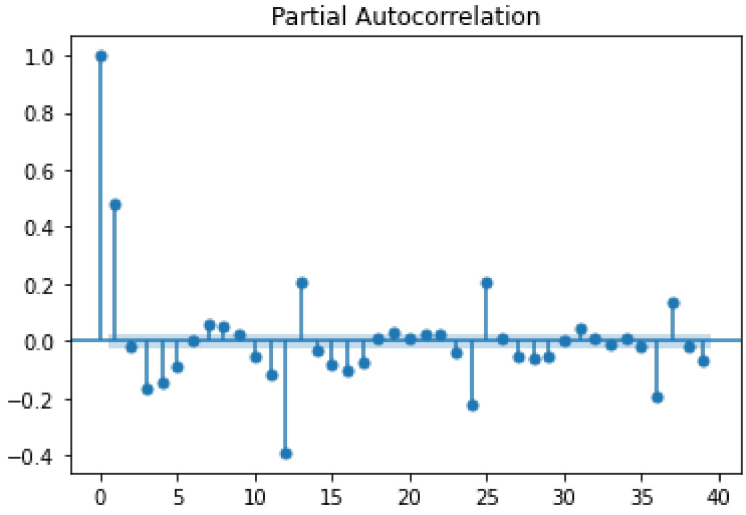
Applying ARIMA algorithm on data

In [73]:
```
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
from statsmodels.graphics.api import qqplot
```

In [74]:
```
plot_acf(df['actual_temp'])
plt.show()
## q=3
```



In [75]:
```
plot_pacf(df['actual_temp'])
plt.show()
##p=2
```



In [76]:
```
df=df[['actual_temp']]
```

In [77]:
```
df.shape
```

Out[77]: (7057, 1)

In [103…
```
training_data=df[0:5000]
testing_data=df[5000:]
```

In [104…
```
from statsmodels.tsa.arima_model import ARIMA
```

In [105…
```
arima=ARIMA(training_data,order=(2,1,3))
```

In [106…
```
model = arima.fit()
```

In [107…
```
pred = model.forecast(steps=len(testing_data))[0]
```

In [108…
```
pred
```

Out[108…
```
array([ 0.77761143, -0.48007918, -1.27074673, ...,  0.02473165,
        0.02473883,  0.024746  ])
```

In [109…
```
len(pred)
```

Out[109…
```
2057
```

In [110…
```
len(testing_data)
```

Out[110…
```
2057
```

In [111…
```
np.sqrt(mean_squared_error(testing_data,pred))
```

Out[111…
```
1.6875915476395478
```

In [ ]:
```
### this proves arima>>MA
```