# Time Series Arima

Monthly Temperature datatset of Takizawa City (JMA Dataset)

In [36]:

Out[36]:

| | Date | Temp |
|---|---|---|
| **0** | 1923/9/1 | 19.2 |
| **1** | 1923/10/1 | 11.8 |
| **2** | 1923/11/1 | 5.8 |
| **3** | 1923/12/1 | -0.3 |
| **4** | 1924/1/1 | -3.3 |

In [37]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1157 entries, 0 to 1156
Data columns (total 2 columns):
Date    1157 non-null object
Temp    1157 non-null float64
dtypes: float64(1), object(1)
memory usage: 18.2+ KB
```

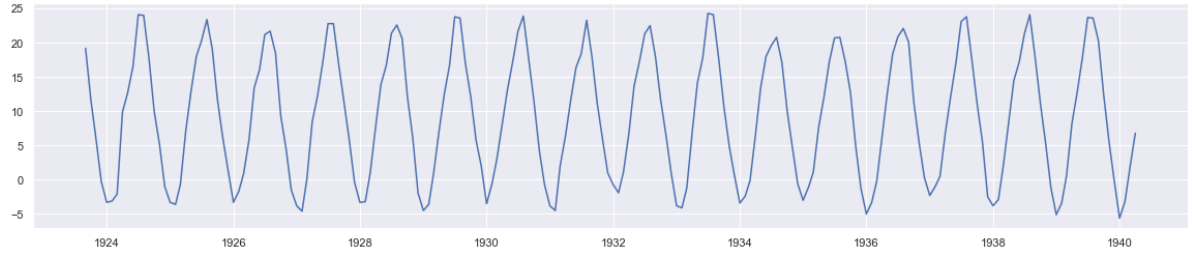
In [38]:

Out[38]:

| | Temp |
|---|---|
| **Date** | |
| **1923-09-01** | 19.2 |
| **1923-10-01** | 11.8 |
| **1923-11-01** | 5.8 |
| **1923-12-01** | -0.3 |
| **1924-01-01** | -3.3 |

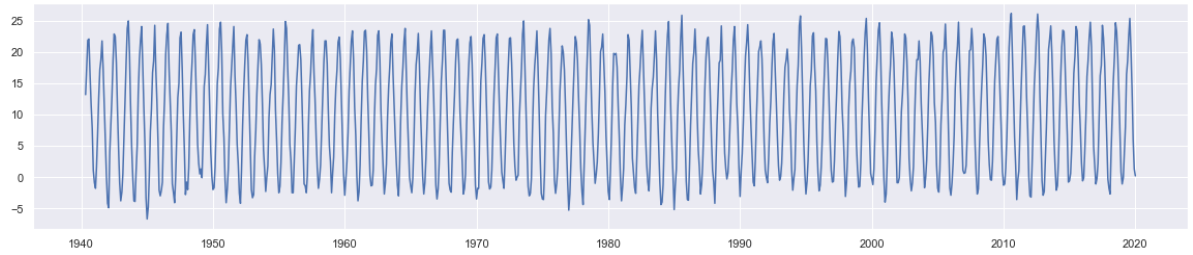
In [39]:

Out[39]: [<matplotlib.lines.Line2D at 0x17f1f8ff588>]



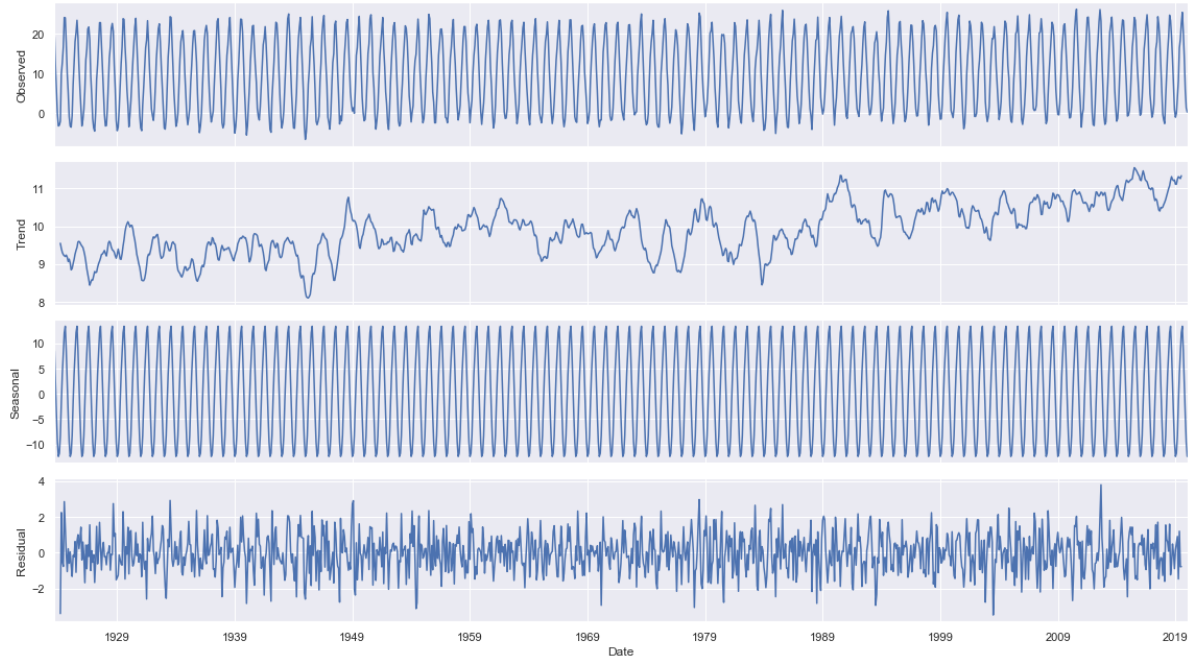
In [40]:

Out[40]: [<matplotlib.lines.Line2D at 0x17f1f505278>]




## Data decomposistion

In [41]:




## Check Stationarity

In [42]:
```
from statsmodels.tsa.stattools import adfuller
def check_stationarity(timeseries):
    result = adfuller(timeseries,autolag='AIC')
    dfoutput = pd.Series(result[0:4], index=['Test Statistic','p-value','#Lags Used','Number of Obs
ervations Used'])
    print('The test statistic: %f' % result[0])
    print('p-value: %f' % result[1])
    print('Critical Values:')
    for key, value in result[4].items():
        print('%s: %.3f' % (key, value))
```

In [43]:
```
check_stationarity(df.Temp)
```

```
The test statistic: -2.949067
p-value: 0.039932
Critical Values:
1%: -3.436
5%: -2.864
10%: -2.568
```

## How to determine AR(p),MA(q) and I(d) values for ARIMA model?

In [44]:
```
df['temperature'] = df['Temp']*(9/5)+32
```

In [45]:
```
import numpy as np
ts_temp_log = np.log(df)
ts_temp_log
```

Out[45]:

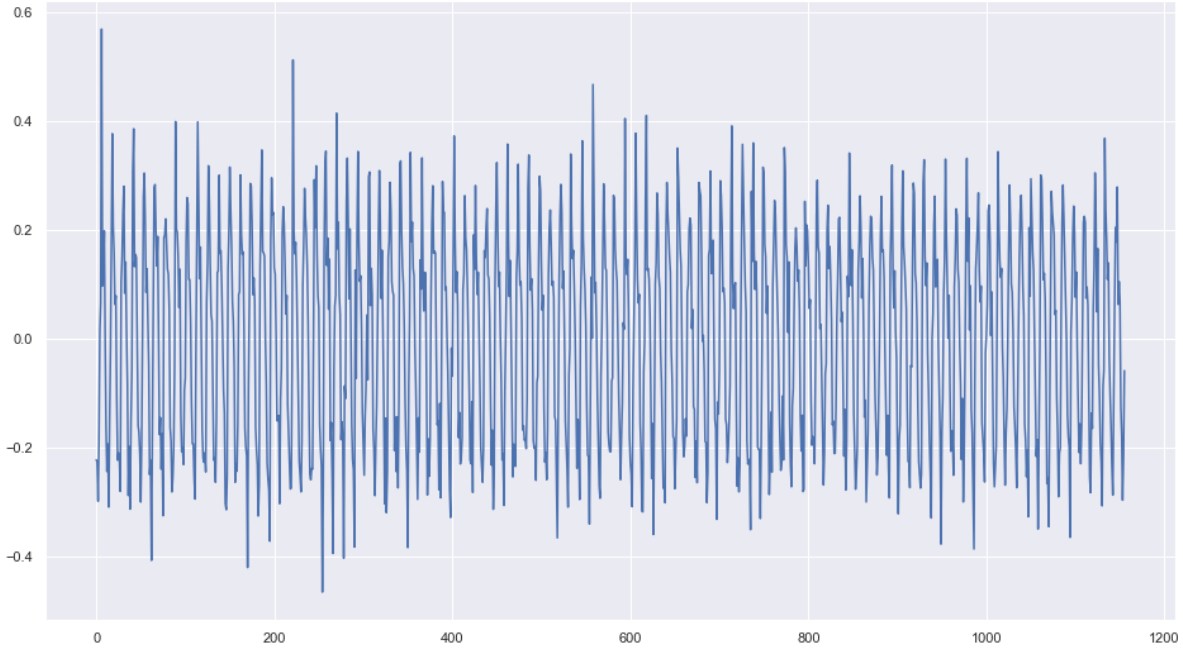
| Date | Temp | temperature |
|---|---|---|
| **1923-09-01** | 2.954910 | 4.198104 |
| **1923-10-01** | 2.468100 | 3.974810 |
| **1923-11-01** | 1.757858 | 3.748091 |
| **1923-12-01** | NaN | 3.448717 |
| **1924-01-01** | NaN | 3.260402 |
| **...** | ... | ... |
| **2019-09-01** | 3.010621 | 4.227418 |
| **2019-10-01** | 2.646175 | 4.049696 |
| **2019-11-01** | 1.774952 | 3.752324 |
| **2019-12-01** | 0.262364 | 3.536311 |
| **2020-01-01** | -1.609438 | 3.476923 |

1157 rows × 2 columns

In [46]:
```
ts_temp_log_diff = np.diff(ts_temp_log.temperature)

plt.plot(ts_temp_log_diff)
```

Out[46]: [<matplotlib.lines.Line2D at 0x17f1f4639e8>]



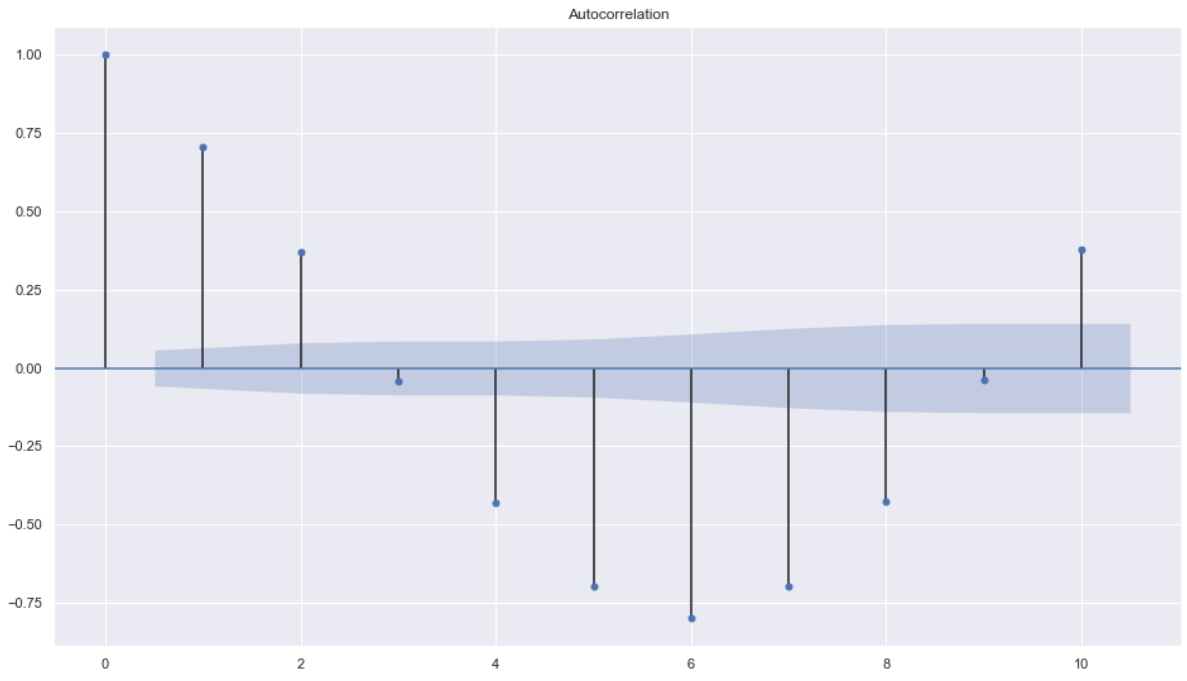
In [47]:
```
check_stationarity(ts_temp_log_diff)
```

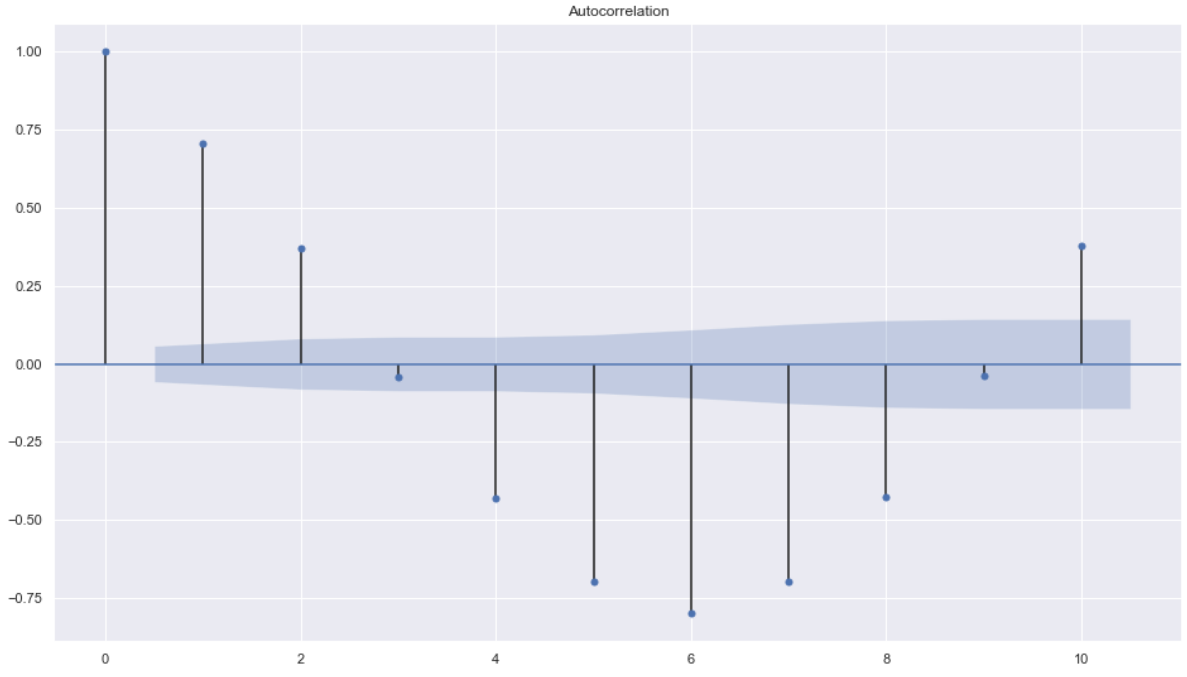
```
The test statistic: -14.356885
p-value: 0.000000
Critical Values:
1%: -3.436
5%: -2.864
10%: -2.568
```

In [48]:
```
## Determine P and Q value from ACF and PACF plot
```

In [49]:
```
from statsmodels.graphics.tsaplots import plot_acf
plot_acf(ts_temp_log_diff, lags=10)
```

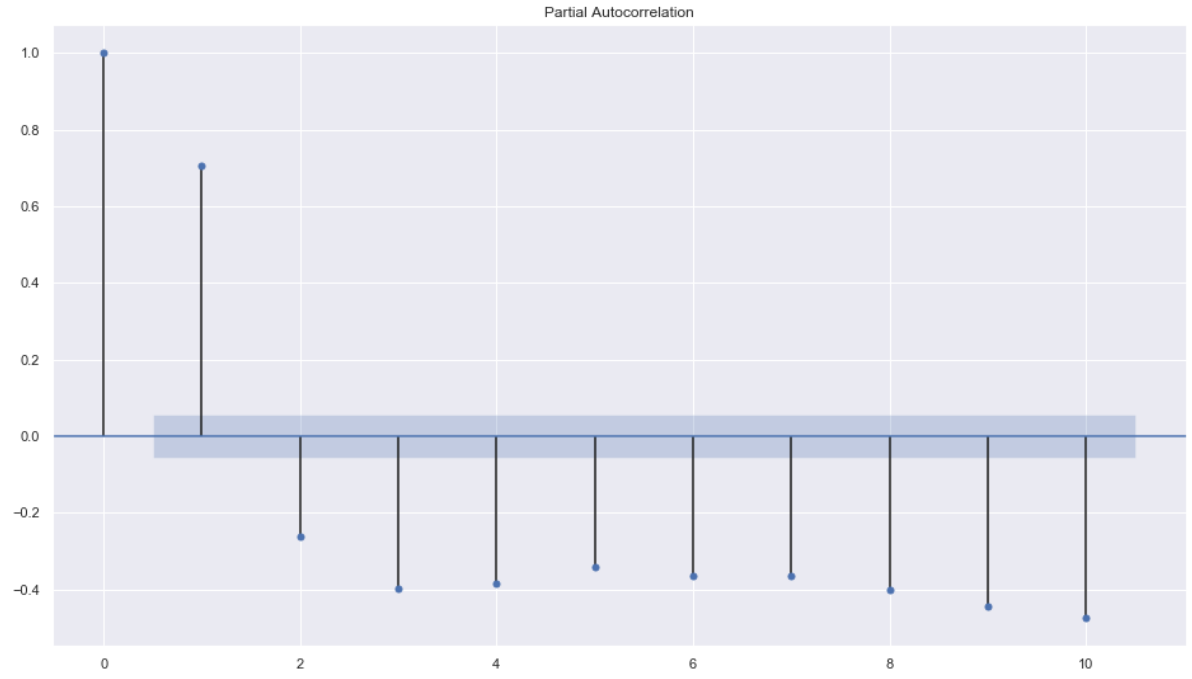
Out[49]:



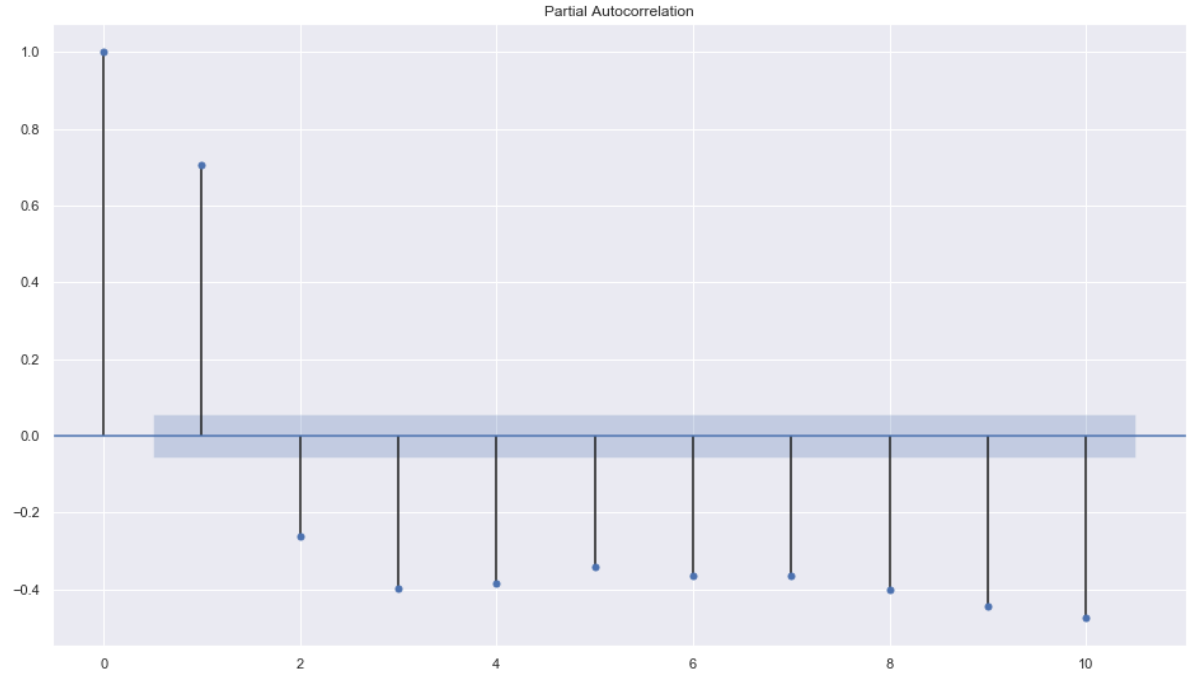



In [50]:
```
from statsmodels.graphics.tsaplots import plot_pacf
plt.figure(figsize=(10,2))
plot_pacf(ts_temp_log_diff,lags=10)
```

Out[50]:




<Figure size 720x144 with 0 Axes>



Once our data is set to stationary then the next task is to determine the appropriate value of ARIMA model i.e. p and q

We can learn some important properties of our time series data with the help of Auto Correlation(ACF) and Partial Auto Correlation (PACF) graphs.

This provide useful descriptive properties for understanding which model can be used for time series forecasting

ACF measures the linear relationships between observations at different time lags.

In other words, ACF is used to understand if there exists a correlation between a time series data point with another point as a function of their time difference

The Partial Auto Correlation factor(PACF) is the partial correlation between the two points at a specific lag of time.

Plotting the partial autocorrelative functions one could determine the appropriate lags p in an AR (p) model or in an extended ARIMA (p,d,q) model

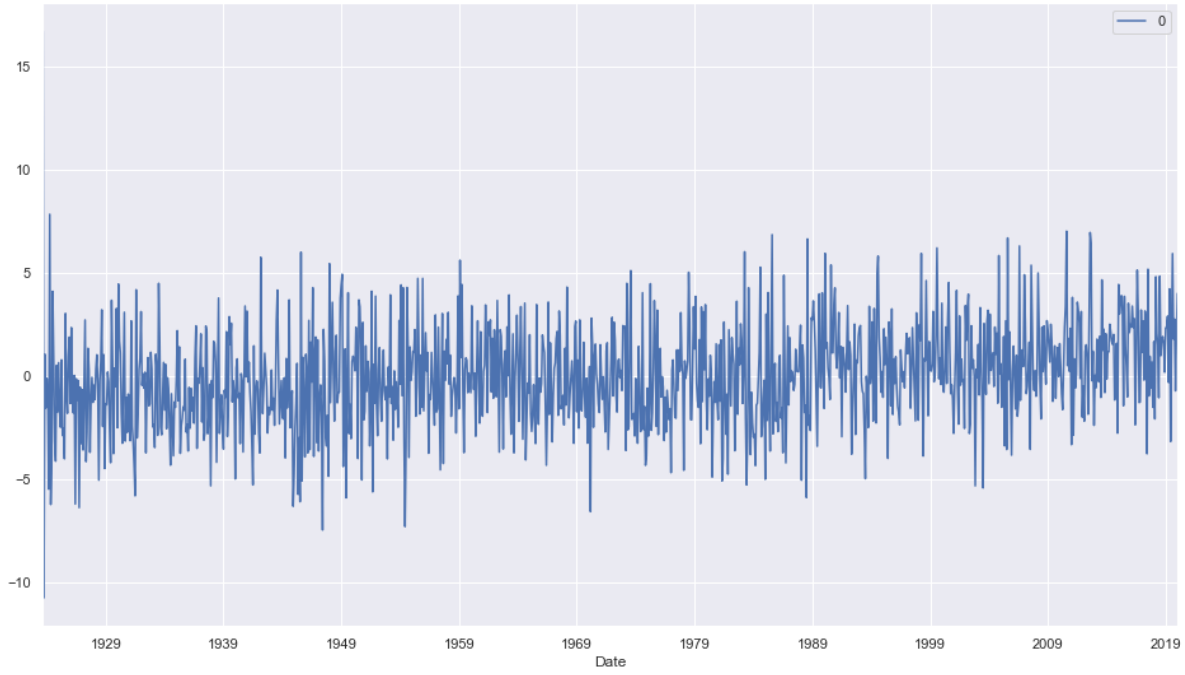
From the PACF plot we can see a significant correlation at lag of 2. So the PACF suggests an AR(2) model. So an inital order for the model will be (2,0,3)

In [51]:
```
from statsmodels.tsa.arima_model import ARIMA
mod = ARIMA(df.temperature,order=(2,0,3))
results = mod.fit()
print(results.summary())
```

```
                              ARMA Model Results
==============================================================================
Dep. Variable:            temperature   No. Observations:                 1157
Model:                 ARMA(2, 3)   Log Likelihood               -2673.115
Method:                    css-mle   S.D. of innovations              2.424
Date:                Wed, 22 Jul 2020   AIC                           5360.229
Time:                        14:39:54   BIC                           5395.604
Sample:                    09-01-1923   HQIC                          5373.579
                         - 01-01-2020
==========================================================================================
                             coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------------------
const                     49.8378      0.001   4.15e+04      0.000      49.835      49.840
ar.L1.temperature          1.7320   3.96e-05   4.38e+04      0.000       1.732       1.732
ar.L2.temperature         -1.0000        nan        nan        nan         nan         nan
ma.L1.temperature         -1.3440      0.027    -49.028      0.000      -1.398      -1.290
ma.L2.temperature          0.3332      0.047      7.046      0.000       0.241       0.426
ma.L3.temperature          0.3772      0.027     13.722      0.000       0.323       0.431
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1            0.8660           -0.5000j            1.0000           -0.0833
AR.2            0.8660           +0.5000j            1.0000            0.0833
MA.1            0.8709           -0.5013j            1.0049           -0.0831
MA.2            0.8709           +0.5013j            1.0049            0.0831
MA.3           -2.6252           -0.0000j            2.6252           -0.5000
-----------------------------------------------------------------------------
```

In [52]:
```
# Residuals Diagnosis
residuals = pd.DataFrame(results.resid)
residuals.plot()
```

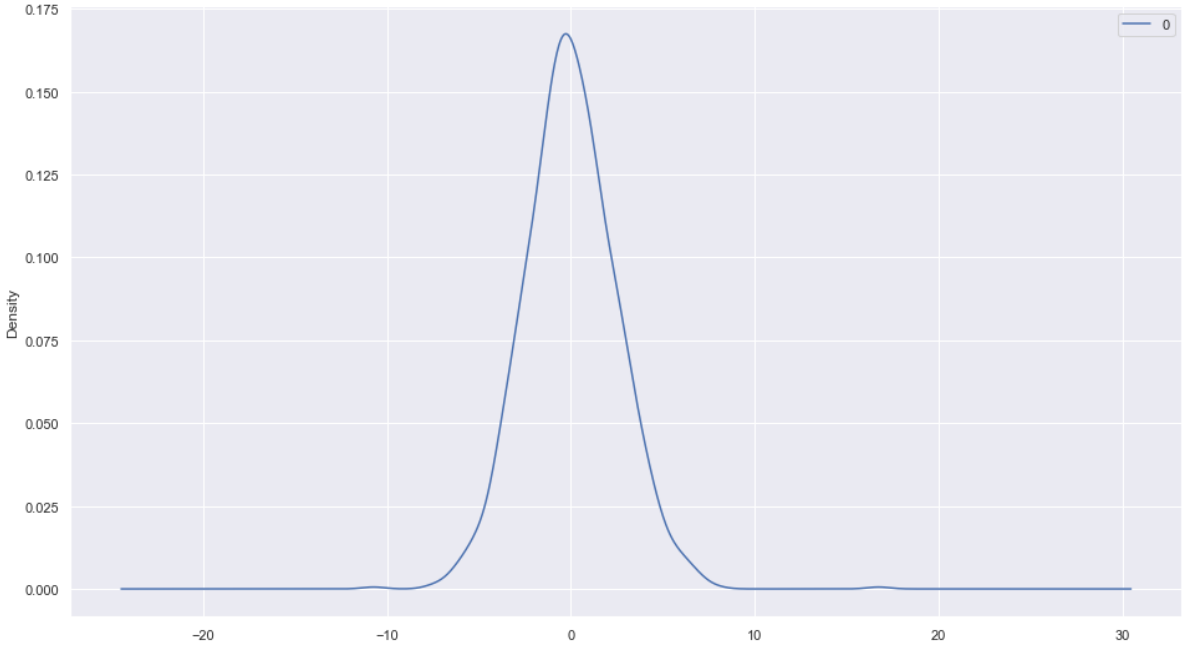
Out[52]: <matplotlib.axes._subplots.AxesSubplot at 0x17f20421e10>




Next we will check if these residuals are normally distributed and looks Gaussian or not. So we will plot the density plot to check this. This looks normal with a long left tail and centered at Zero

In [53]:
```
residuals.plot(kind='kde')
```

Out[53]: <matplotlib.axes._subplots.AxesSubplot at 0x17f204da2e8>



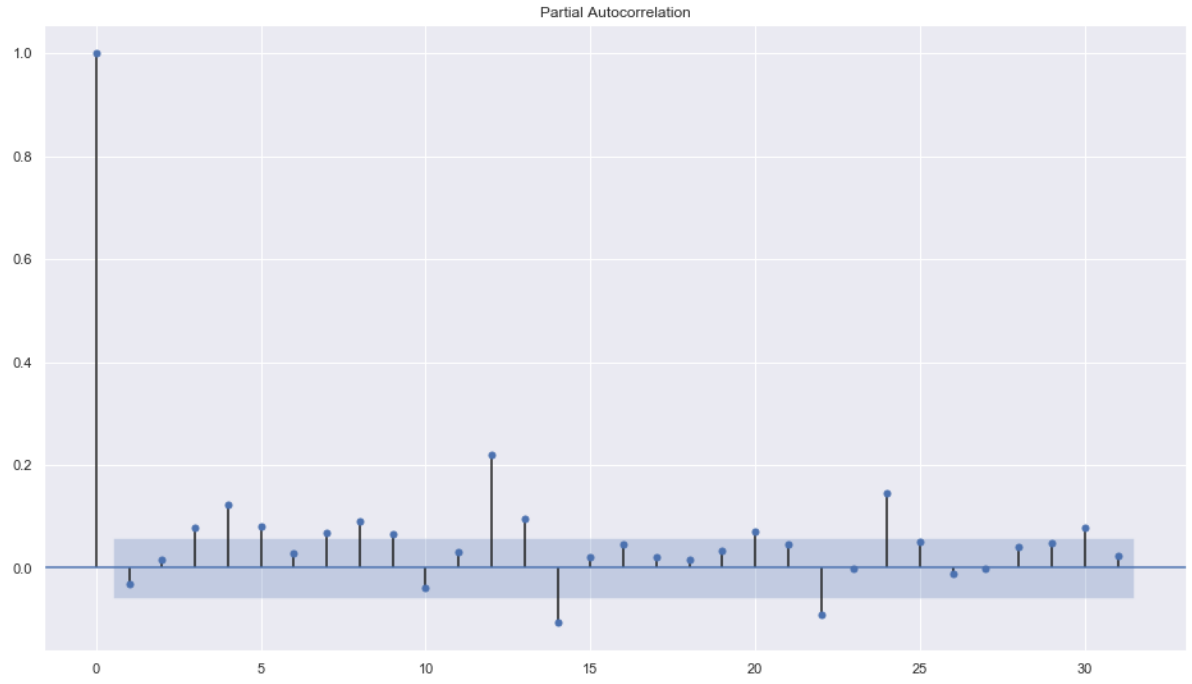
In [54]:
```
residuals.describe()
```

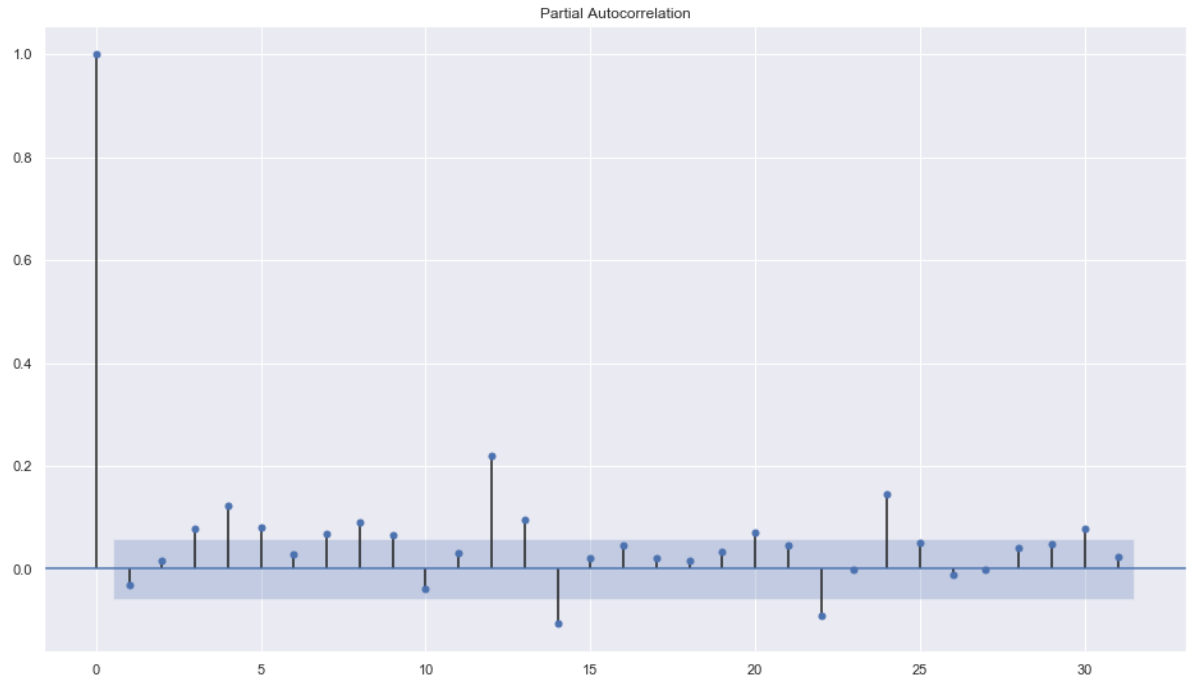
Out[54]:

| | 0 |
|---|---|
| **count** | 1157.000000 |
| **mean** | 0.004707 |
| **std** | 2.501138 |
| **min** | -10.731751 |
| **25%** | -1.585819 |
| **50%** | -0.086005 |
| **75%** | 1.564582 |
| **max** | 16.722210 |

In [55]:
```
plot_pacf(residuals)
```

Out[55]:

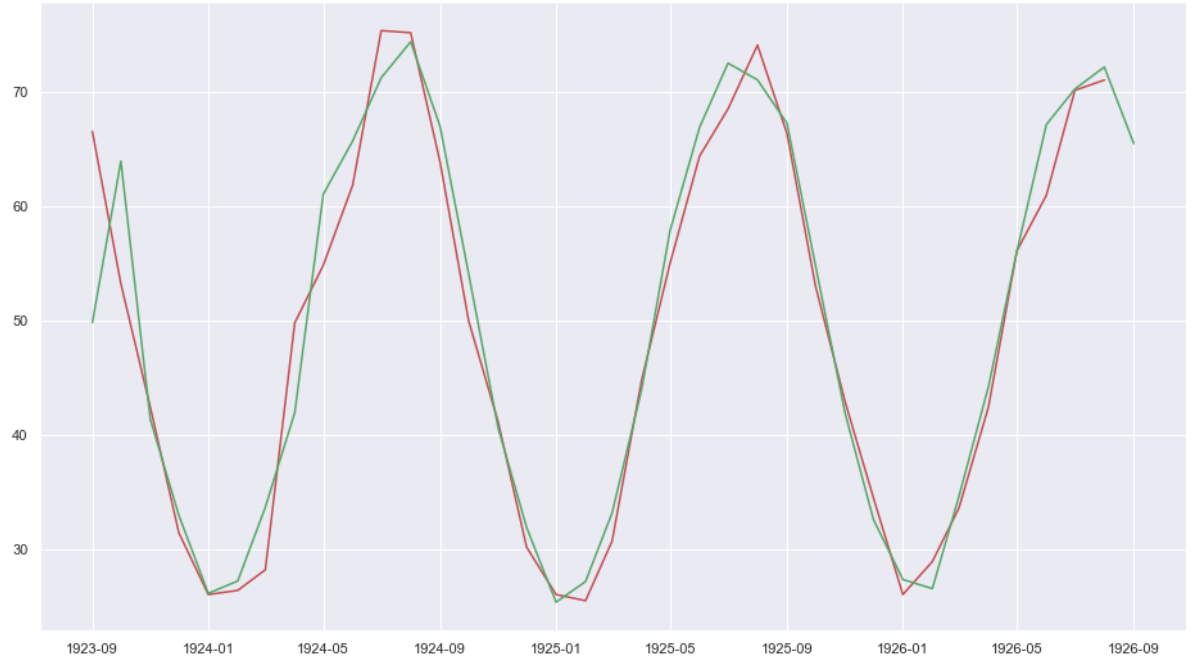


# ARIMA predict

In [56]:
```
from math import sqrt
from sklearn.metrics import mean_squared_error
plt.plot(df.temperature[:36],color='r')
plt.plot(results.predict(0,36),color='g')
```

Out[56]: [<matplotlib.lines.Line2D at 0x17f1fabb390>]




In [57]:
```
rmse =sqrt(mean_squared_error(df.temperature, results.predict()))
print(rmse)
```

2.50006175088718

## Forecast

Next and Final Step to forecast the temperature of upcoming years i.e. Out of Sample Forecast. We will use the forecast method of statsmodel for this task.

You can pass steps as one of the parameter i.e. number of out of sample forecasts from the end of the sample

And can provide the confidence interval(alpha) for your forecast too

This function will return following three values:

```
Forecast : Array of Out of Sample forecast
stderr : Array of the standard error of the forecasts
confidence Interval : 2d array of the confidence interval for the forecast
```

We are forecasting the temperature for next 3 years i.e. 36 months so our steps will be 36 and for a confidence interval of 95% we will pass the alpha value as 0.05

In [58]:
```
n=36
forecast,err,ci = results.forecast(steps=n, alpha=0.05)
df_forecast = pd.DataFrame({'forecast':forecast},index=pd.date_range(start='1/1/2020', periods=n, f
req='MS'))
```

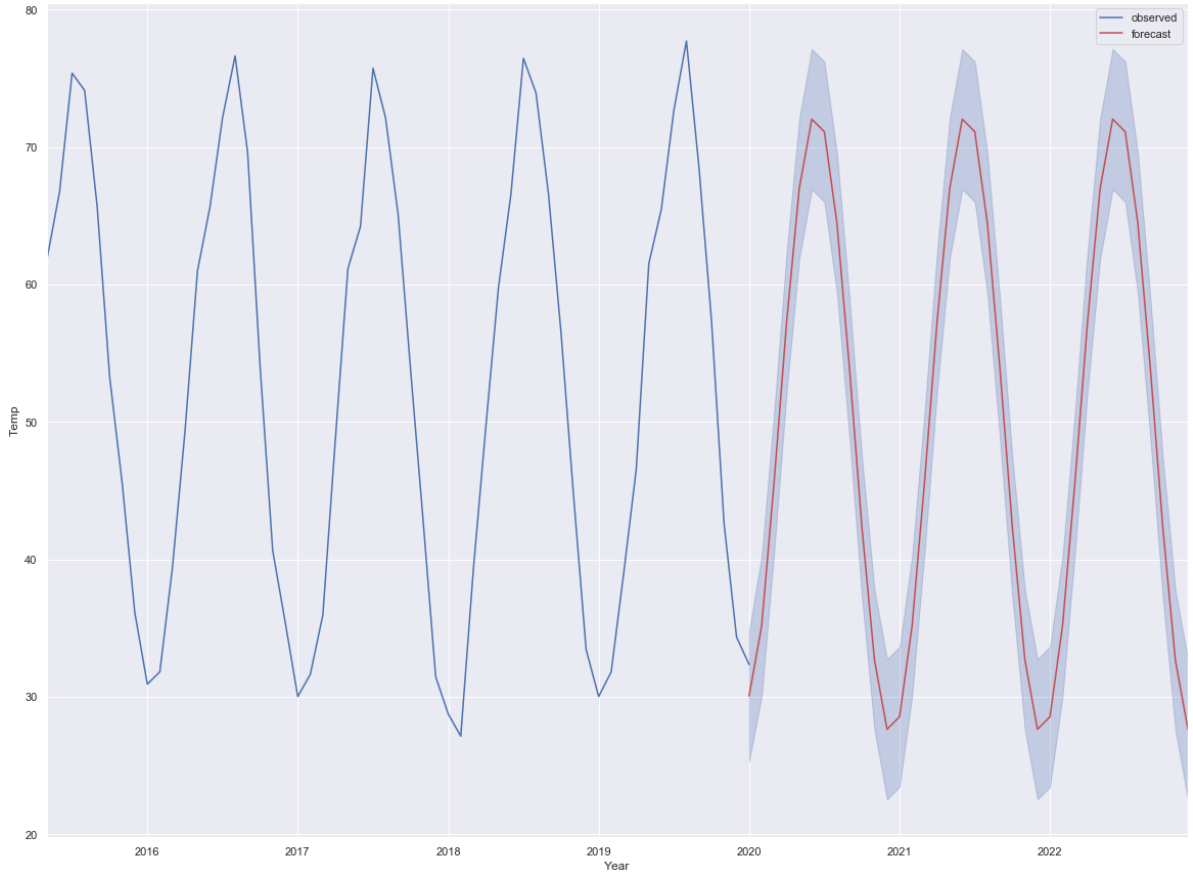
In [59]:
```
df_forecast.tail()
```

Out[59]:

| | forecast |
|---|---|
| **2022-08-01** | 64.497464 |
| **2022-09-01** | 53.948810 |
| **2022-10-01** | 42.298511 |
| **2022-11-01** | 32.668642 |
| **2022-12-01** | 27.639828 |

In [60]:
```
ax = df[1100:].temperature.plot(label='observed', figsize=(20, 15))
df_forecast.plot(ax=ax,label='Forecast',color='r')
ax.fill_between(df_forecast.index,
                ci[:,0],
                ci[:,1], color='b', alpha=.25)
ax.set_xlabel('Year')
ax.set_ylabel('Temp')

plt.legend()
plt.show()
```




In [61]:
```
#fahrenheit to degree
df_forecast['Degree']=(df_forecast['forecast']-32)*5/9
df_forecast.head()
```

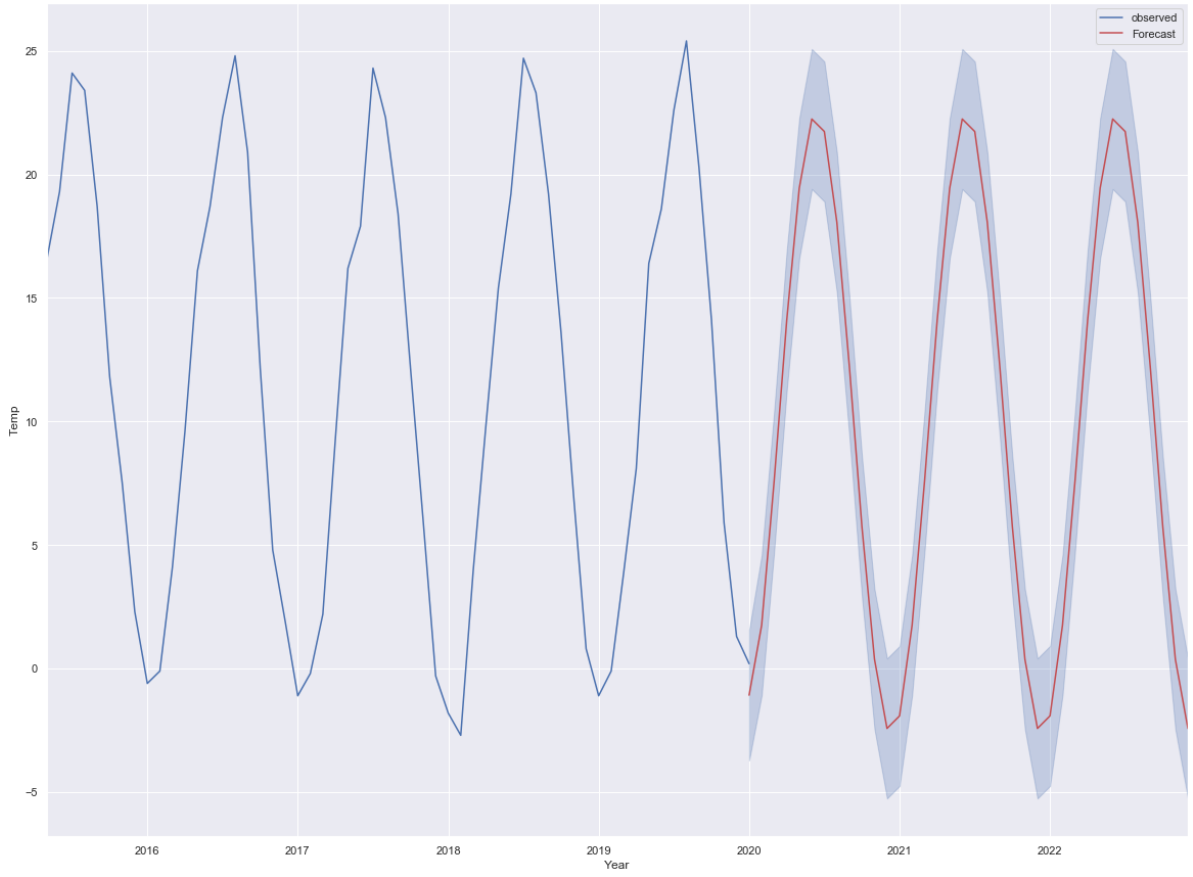
Out[61]:

| | forecast | Degree |
|---|---|---|
| **2020-01-01** | 30.064895 | -1.075058 |
| **2020-02-01** | 35.160250 | 1.755694 |
| **2020-03-01** | 45.704395 | 7.613553 |
| **2020-04-01** | 57.356181 | 14.086767 |
| **2020-05-01** | 66.993135 | 19.440631 |

In [ ]:

In [62]:
```
ax = df[1100:].Temp.plot(label='observed', figsize=(20, 15))
df_forecast['Degree'].plot(ax=ax,label='Forecast',color='r')
ax.fill_between(df_forecast.index,
                (ci[:,0]-32)*5/9,
                (ci[:,1]-32)*5/9, color='b', alpha=.25)
ax.set_xlabel('Year')
ax.set_ylabel('Temp')

plt.legend()
plt.show()
```




In [63]: