Q1 a)

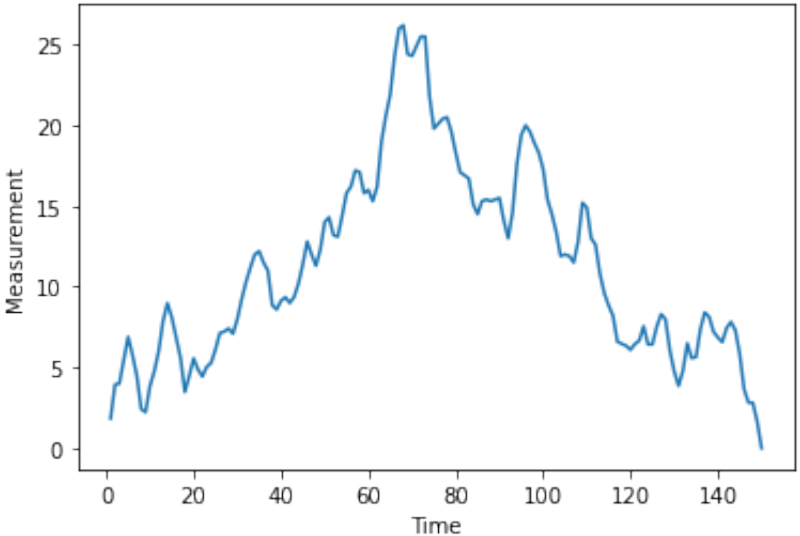
In [21]:
```
import pandas as pd
from pandas import Series
import numpy as np
import matplotlib.pyplot as plt
measurement = pd.read_csv('Measurement_Q1.csv', names = ["time", "measure
measurement
```

Out[21]:

	time	measurement
0	1	1.84
1	2	3.93
2	3	4.00
3	4	5.42
4	5	6.89
...	...	...
145	146	3.65
146	147	2.82
147	148	2.81
148	149	1.64
149	150	0.00

150 rows × 2 columns

In [2]:
```
x = measurement.time
y = measurement.measurement
plt.plot (x, y)
plt.xlabel ('Time')
plt.ylabel ('Measurement')
plt.show()
```



In [3]:
```
import statsmodels.api as sm
```

In [4]:
```
measurement_model = sm.tsa.arima.ARIMA(measurement.measurement, order=(0,
```

In [5]:
```
output = measurement_model.fit()
```

In [6]:
```
output.summary()
```

Out[6]:

SARIMAX Results

Dep. Variable:	measurement	**No. Observations:**	150
Model:	ARIMA(0, 1, 1)	**Log Likelihood**	-202.609
Date:	Thu, 13 Oct 2022	**AIC**	409.217
Time:	12:54:06	**BIC**	415.225
Sample:	0	**HQIC**	411.658
	- 150		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ma.L1	0.7531	0.060	12.573	0.000	0.636	0.871
sigma2	0.8834	0.109	8.080	0.000	0.669	1.098

Ljung-Box (L1) (Q):	0.26	**Jarque-Bera (JB):**	1.10
Prob(Q):	0.61	**Prob(JB):**	0.58
Heteroskedasticity (H):	1.00	**Skew:**	-0.21
Prob(H) (two-sided):	0.99	**Kurtosis:**	2.98

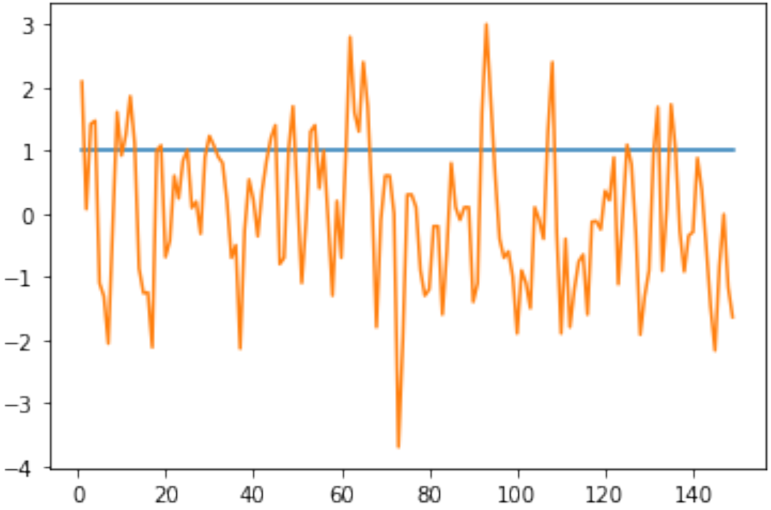
Warnings:

[1] Covariance matrix calculated using the outer product of gradients (complex-step).

Q1b)

In [7]:
```
measurement2=measurement.diff()
plt.plot(measurement2)
```

Out[7]:
```
[<matplotlib.lines.Line2D at 0x7f9f50138df0>,
 <matplotlib.lines.Line2D at 0x7f9f50138d90>]
```



Q1 c)

In [8]:
```
measurement_diff = measurement2.iloc[1:]
measurement_model2 = sm.tsa.arima.ARIMA(measurement_diff.measurement, ord
```

In [9]:
```
measurement_model_fit2 = measurement_model2.fit()
```

In [10]:
```
measurement_model_fit2.summary()
```

Out[10]:

SARIMAX Results

Dep. Variable:	measurement	**No. Observations:**	149
Model:	ARIMA(0, 0, 1)	**Log Likelihood**	-202.607
Date:	Thu, 13 Oct 2022	**AIC**	411.215
Time:	12:54:06	**BIC**	420.227
Sample:	0	**HQIC**	414.876
	- 149		

Covariance Type:	opg

	coef	std err	z	P>|z|	[0.025	0.975]
const	-0.0064	0.137	-0.047	0.962	-0.274	0.261
ma.L1	0.7531	0.061	12.392	0.000	0.634	0.872
sigma2	0.8834	0.110	8.052	0.000	0.668	1.098

Ljung-Box (L1) (Q):	0.26	**Jarque-Bera (JB):**	1.10
Prob(Q):	0.61	**Prob(JB):**	0.58
Heteroskedasticity (H):	0.99	**Skew:**	-0.21
Prob(H) (two-sided):	0.98	**Kurtosis:**	2.98

Warnings:

[1] Covariance matrix calculated using the outer product of gradients (complex-step).

Q1 d)

After implementing all parts above it can be observed that both methods give basically the same result because they are doing the same thing. a) part is more of a direct approach whereas, b) is a longer process. We can oberserve one more thing that is AIC which tells the better fit for a model. The lower the AIC better the model will fit. Therefore, a) has lower AIC means it is likely that the model will fit better through this method.

Q2 a)

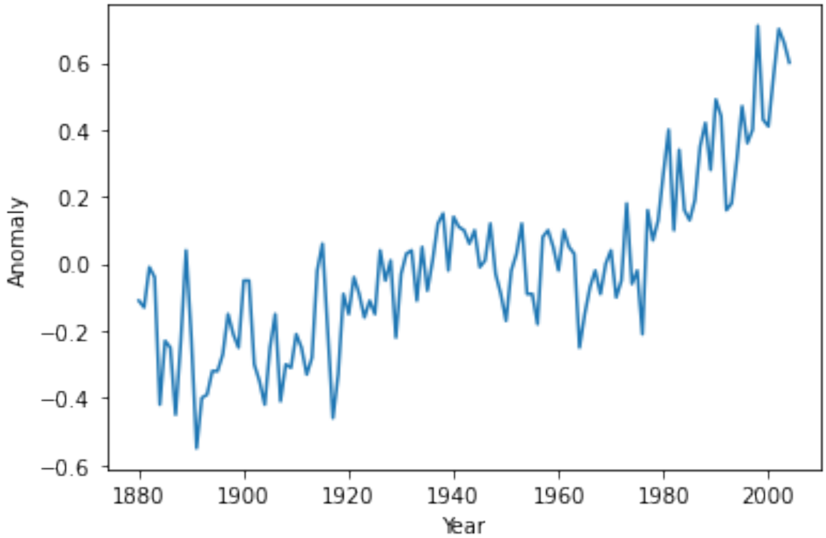
In [11]:
```
df=pd.read_csv('GlobalAirTemperature.csv')
df.head()
```

Out[11]:

	Year	Anomaly, C
0	1880	-0.11
1	1881	-0.13
2	1882	-0.01
3	1883	-0.04
4	1884	-0.42

In [12]:

```
x = df['Year']
y = df['Anomaly, C']
plt.plot (x, y)
plt.xlabel ('Year')
plt.ylabel ('Anomaly')
plt.show()
```



In [13]:
```
anomaly_model = sm.tsa.arima.ARIMA(df['Anomaly, C'], order=(0,1,1))
```

In [14]:
```
output2=anomaly_model.fit()
```

In [15]:
```
output2
```

Out[15]: <statsmodels.tsa.arima.model.ARIMAResultsWrapper at 0x7f9f70ce7760>

In [16]:
```
output2.summary()
```

Out[16]:

SARIMAX Results

Dep. Variable:	Anomaly, C	**No. Observations:**	125
Model:	ARIMA(0, 1, 1)	**Log Likelihood**	72.910
Date:	Thu, 13 Oct 2022	**AIC**	-141.821
Time:	12:54:06	**BIC**	-136.180
Sample:	0	**HQIC**	-139.530
	- 125		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ma.L1	-0.6640	0.072	-9.243	0.000	-0.805	-0.523
sigma2	0.0180	0.002	8.182	0.000	0.014	0.022

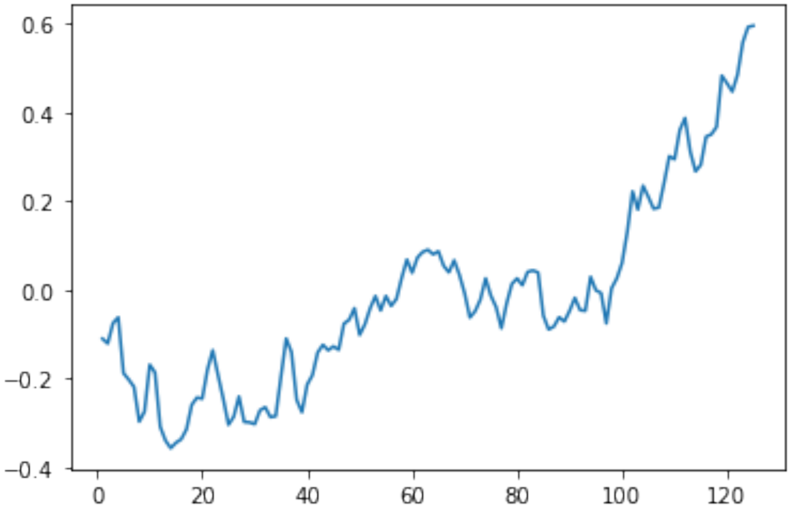
Ljung-Box (L1) (Q):	2.36	**Jarque-Bera (JB):**	2.20
Prob(Q):	0.12	**Prob(JB):**	0.33
Heteroskedasticity (H):	0.87	**Skew:**	-0.29
Prob(H) (two-sided):	0.65	**Kurtosis:**	3.32

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

In [17]:
```
anomaly_predict = output2.predict (1, len(y), typ = 'levels').rename("Pre
```

In [18]:
```
plt.plot (anomaly_predict)
```

Out[18]:
```
[<matplotlib.lines.Line2D at 0x7f9f61ef4ac0>]
```



In [19]:

```
from sklearn.metrics import mean_squared_error
mse1 = mean_squared_error (y, anomaly_predict)
mse1
```

Out[19]:

```
0.007848867110038906
```

In [22]:
```
sse=np.sum((y-anomaly_predict)**2)
sse
```

Out[22]:
```
2.2339840576983625
```

Q2 b)

In [23]:
```
anomaly_model2 = sm.tsa.arima.ARIMA(df['Anomaly, C'], order=(0,1,2))
```

In [24]:
```
output_ima2 = anomaly_model2.fit()
```

In [25]:
```
output_ima2.summary()
```

Out[25]:

SARIMAX Results

Dep. Variable:	Anomaly, C	**No. Observations:**	125
Model:	ARIMA(0, 1, 2)	**Log Likelihood**	76.508
Date:	Thu, 13 Oct 2022	**AIC**	-147.016

Time:	12:55:29	**BIC**	-138.555
Sample:	0	**HQIC**	-143.579
	- 125		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ma.L1	-0.4676	0.099	-4.736	0.000	-0.661	-0.274
ma.L2	-0.2296	0.090	-2.565	0.010	-0.405	-0.054
sigma2	0.0170	0.002	7.631	0.000	0.013	0.021

Ljung-Box (L1) (Q):	0.13	**Jarque-Bera (JB):**	1.18
Prob(Q):	0.72	**Prob(JB):**	0.55
Heteroskedasticity (H):	1.04	**Skew:**	-0.23
Prob(H) (two-sided):	0.91	**Kurtosis:**	3.12

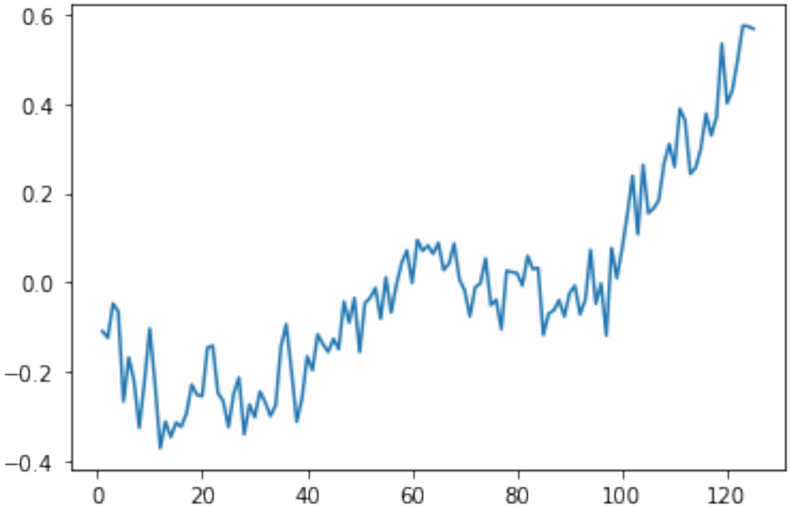
Warnings:

[1] Covariance matrix calculated using the outer product of gradients (complex-step).

In [26]:
```
anomaly_predict2 = output_ima2.predict (1, len(y), typ = 'levels').rename
```

In [27]:
```
plt.plot(anomaly_predict2)
```

Out[27]:
```
[<matplotlib.lines.Line2D at 0x7f9f710b0f40>]
```



In [28]:
```
mse2 = mean_squared_error (y, anomaly_predict2)
mse2
```

Out[28]:
```
0.0044938554865248
```

In [29]:
```
sse=np.sum((y-anomaly_predict2)**2)
sse
```

Out[29]:
```
2.1120405362089145
```

Q2 c)

After implementing IMA(1,1) and IMA(1,2) it was observed that sse for order 2 was less, therefore, the model was more close to orignal values than order 1. Also AIC value was less for order 2 and from above explaination we know that lower AIC is better. Ultimately, we can conclude that for this data IMA(1,2) would be a better fit.

Q3 a)

In [30]:
```
dfmeasure = pd.read_csv('Measurement_Q3.csv')
dfmeasure.head()
```

Out[30]:

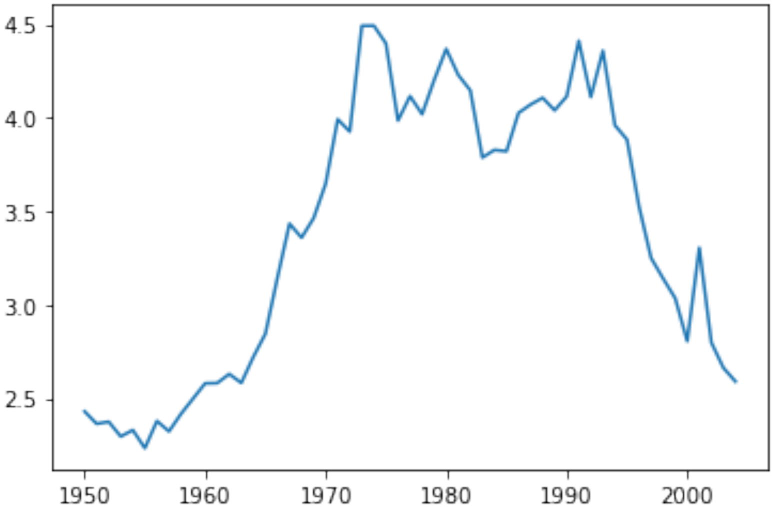
	Year	Measurement
0	1950	2.429415
1	1951	2.363364
2	1952	2.374305
3	1953	2.295520
4	1954	2.329716

In [31]:

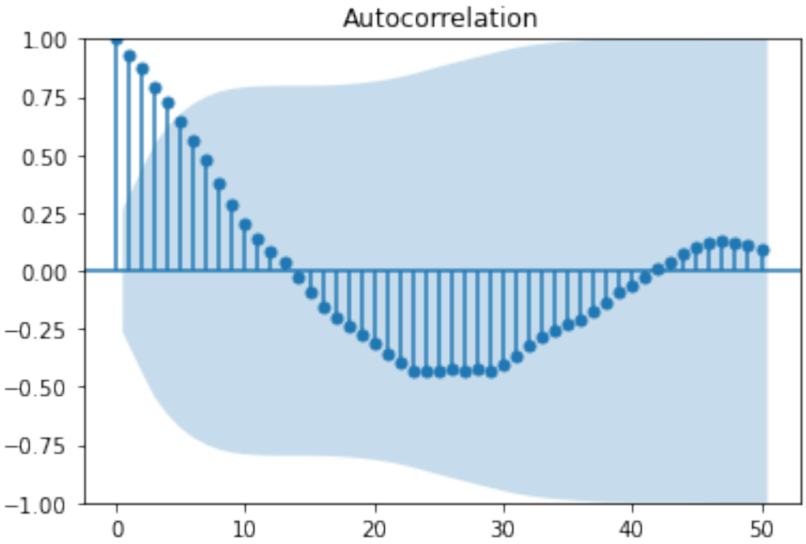
```
x = dfmeasure.Year
y = dfmeasure.Measurement
plt.plot(x,y)
```

Out[31]:

```
[<matplotlib.lines.Line2D at 0x7f9f50249ac0>]
```



In [50]:
```
from statsmodels.graphics.tsaplots import plot_acf
dfmeasure2 = dfmeasure[['Year', 'Measurement']].set_index(['Year'])
plot_acf(dfmeasure2,lags=50);
```



In [51]:
```
dfmeasure2
```

Out[51]:

	Measurement
Year	
1950	2.429415

1951	2.363364
1952	2.374305
1953	2.295520
1954	2.329716
1955	2.233017
1956	2.378179
1957	2.322671
1958	2.416556
1959	2.498199
1960	2.579453
1961	2.580840
1962	2.629293
1963	2.581853
1964	2.720940

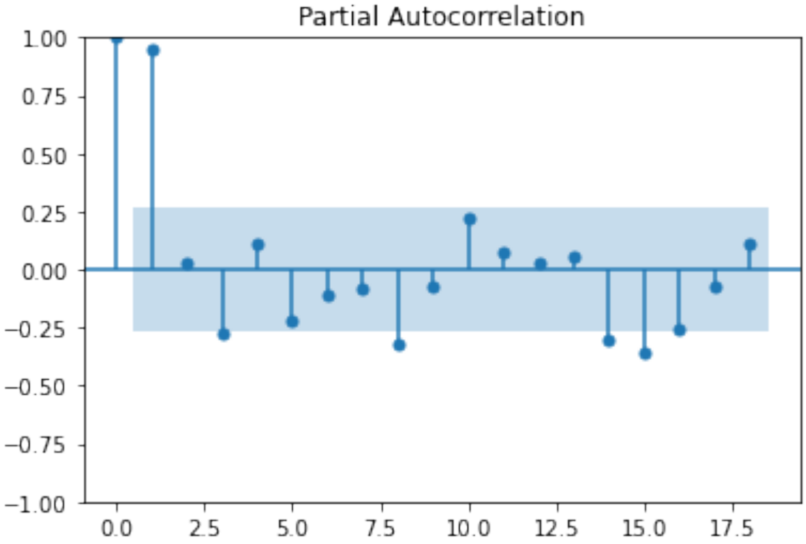
1965	2.844774
1966	3.144862
1967	3.433044
1968	3.358418
1969	3.462620
1970	3.647342
1971	3.991080
1972	3.925702
1973	4.490962
1974	4.491541
1975	4.396567
1976	3.984491
1977	4.115111
1978	4.018538

1979	4.201107
1980	4.367459
1981	4.228103
1982	4.145889
1983	3.786691
1984	3.827373
1985	3.820376
1986	4.025134
1987	4.070130
1988	4.105920
1989	4.039027
1990	4.113978
1991	4.410670
1992	4.110586

1993	4.357700
1994	3.959040
1995	3.882907
1996	3.524803
1997	3.249564
1998	3.139884
1999	3.034263
2000	2.805041
2001	3.304467
2002	2.797697
2003	2.662227
2004	2.589383

In [36]:
```
from statsmodels.graphics.tsaplots import plot_pacf
plot_pacf(dfmeasure2);
```

```
/Users/dhavalgarg/opt/anaconda3/lib/python3.9/site-packages/statsmodels/g
raphics/tsaplots.py:348: FutureWarning: The default method 'yw' can produ
ce PACF values outside of the [-1,1] interval. After 0.13, the default wi
ll change tounadjusted Yule-Walker ('ywm'). You can use this method now b
y setting method='ywm'.
  warnings.warn(
```



Q3 b)

In [52]:
```
me = diff(dfmeasure2)
```

In [54]:
```
me
```

Out[54]:

	Measurement
Year	
1951	-0.066051
1952	0.010941
1953	-0.078785
1954	0.034196
1955	-0.096699
1956	0.145162
1957	-0.055508

1958	0.093885
1959	0.081643
1960	0.081254
1961	0.001387
1962	0.048453
1963	-0.047440
1964	0.139087
1965	0.123834
1966	0.300088
1967	0.288182
1968	-0.074626
1969	0.104202
1970	0.184722
1971	0.343738

1972	-0.065378
1973	0.565260
1974	0.000579
1975	-0.094974
1976	-0.412076
1977	0.130620
1978	-0.096573
1979	0.182569
1980	0.166352
1981	-0.139356
1982	-0.082214
1983	-0.359198
1984	0.040682
1985	-0.006997

1986	0.204758
1987	0.044996
1988	0.035790
1989	-0.066893
1990	0.074951
1991	0.296692
1992	-0.300084
1993	0.247114
1994	-0.398660
1995	-0.076133
1996	-0.358104
1997	-0.275239
1998	-0.109680
1999	-0.105621

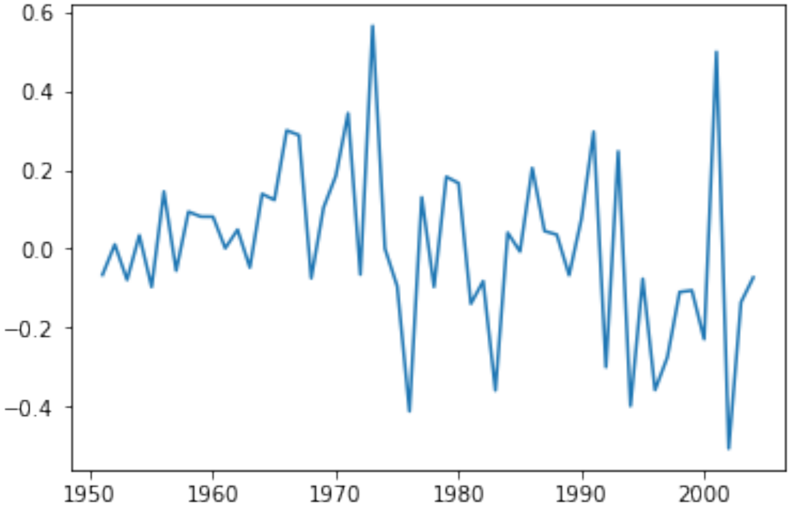
2000	-0.229222
2001	0.499426
2002	-0.506770
2003	-0.135470
2004	-0.072844

In [53]:
```
from statsmodels.tsa.statespace.tools import diff
# measurement2= dfmeasure.Measurement.diff()
```

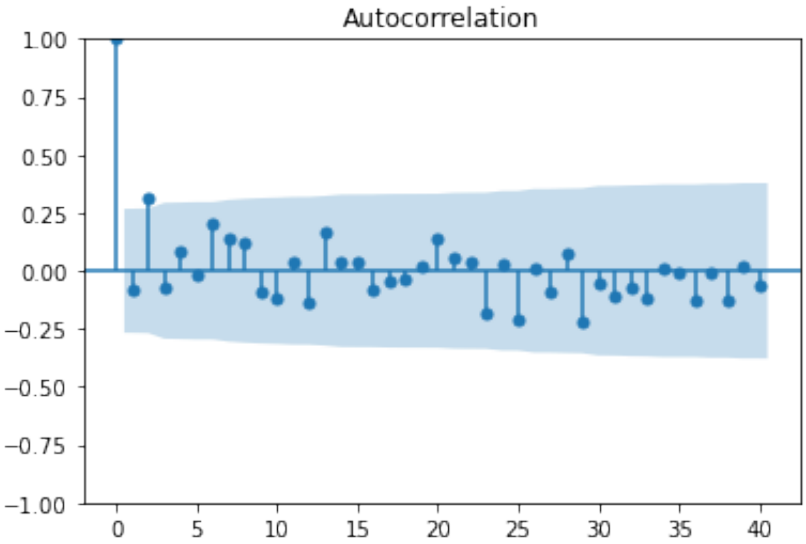
Out[53]:

	Year	Measurement
1	1.0	-0.066051
2	1.0	0.010941
3	1.0	-0.078785
4	1.0	0.034196
5	1.0	-0.096699

In [55]:
```
plt.plot (me)
plt.show()
```

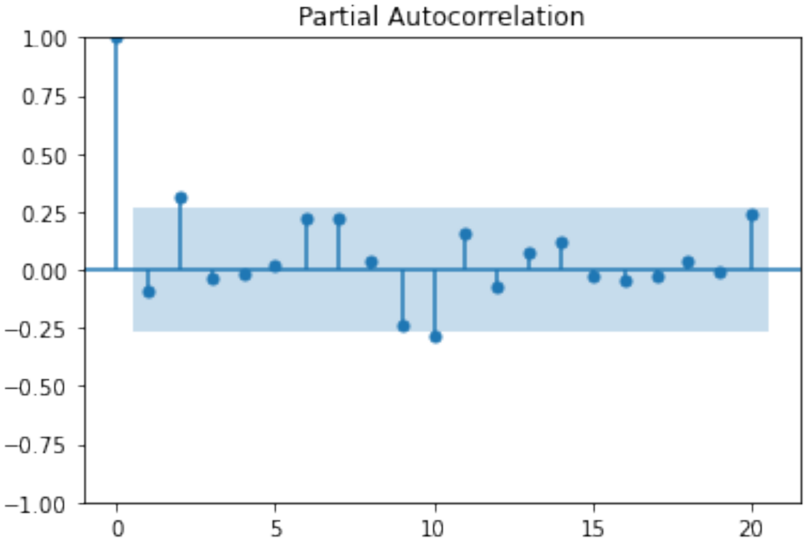


In [58]:
```
# pd.plotting.autocorrelation_plot(measurement2[1:])
dfmeasure3 = measurement_diff[['time', 'measurement']].set_index(['time']
plot_acf(me,lags=40);
```



In [59]:
```
plot_pacf(me,lags=20);
```

```
/Users/dhavalgarg/opt/anaconda3/lib/python3.9/site-packages/statsmodels/g
raphics/tsaplots.py:348: FutureWarning: The default method 'yw' can produ
ce PACF values outside of the [-1,1] interval. After 0.13, the default wi
ll change tounadjusted Yule-Walker ('ywm'). You can use this method now b
y setting method='ywm'.
  warnings.warn(
```



Q3 c)

We cannot tell the order of the model just by looking at the ACF and PACF plots. We can determine that the data will use ARMA model.

One way of finding the order p,q can be to build models for different values of p,q and compare the errors as well as AIC values.The lower the AIC value the better the model is.

In [ ]:

```

```