Q6. To ensure service quality in emergence department, a hospital manager likes to have a strong truthabout patient arrival pattern so that he can plan sufficient resources in advance. He is looking foran accurate forecast tool for patient arrival prediction. The historical data is given in the attachedExcel file. The last 10 days are used for forecast model validation. (1) Plot ACF and PACF with lagk= 1 – 20. What do you observe from the historical patientarrivals, e.g, trend, seasonality, noise, etc? (2) Develop your forecast model ARIMA(p, d, q). Elaborate your determination for the best settings.

(1)

In [154]:
```
#pip install pmdarima
```

```
  File "<ipython-input-154-8ff61c928b9f>", line 2
    pip install Pandoc
               ^
SyntaxError: invalid syntax
```

In [123]:
```
import pandas as pd
import numpy as np
import matplotlib.pylab as plt
import statsmodels.api as sm
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
from pmdarima import auto_arima
from statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.tsa.arima_model import ARIMA
from sklearn.metrics import mean_squared_error
from statsmodels.tools.eval_measures import mse, rmse, meanabs
```

In [62]:
```
df = pd.read_csv('Google Drive/IE5400_HW02_Q6 data.csv')
df['Time'] = np.arange(len(df))
df
```

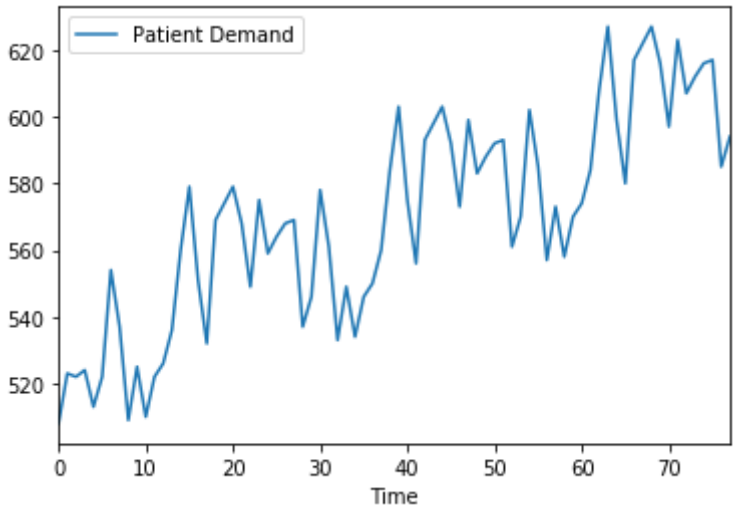
Out[62]:

	Patient Demand	Time
0	508	0
1	523	1
2	522	2
3	524	3
4	513	4
...	...	...
73	612	73
74	616	74
75	617	75
76	585	76
77	594	77

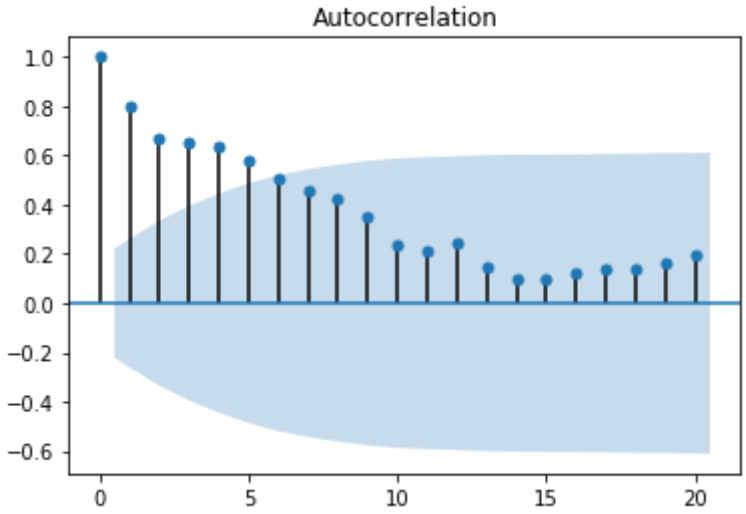
78 rows × 2 columns

In [63]:
```
df.set_index('Time', inplace=True)
df.plot()
```

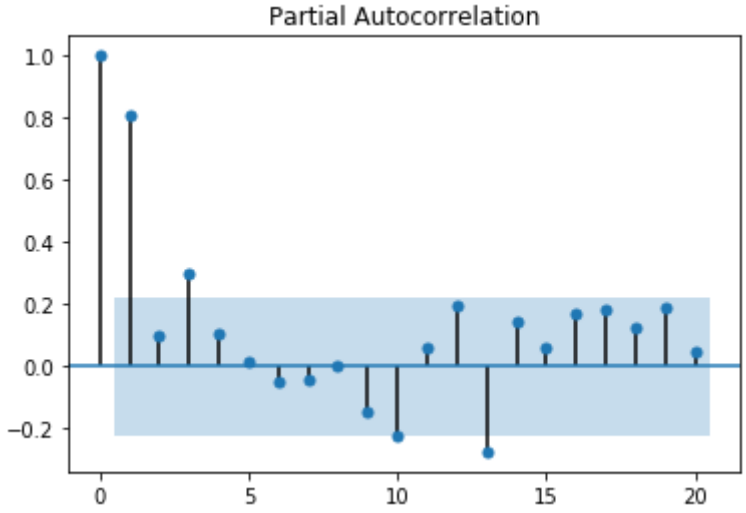
Out[63]: <matplotlib.axes._subplots.AxesSubplot at 0x7f9765e6bb10>



In [50]:
```
sm.graphics.tsa.plot_acf(df.values.squeeze(), lags=20)
plt.show()
```



In [51]:
```
sm.graphics.tsa.plot_pacf(df.values.squeeze(), lags=20)
plt.show()
```



Trend: From the time series plot, we see a positive trend with noise and fluctuations with time.

Seasonality: From ACF and PACF, we find small seasonal effect since lags from ACF & PACF plots aredescent first and then going up a litte. Significant lags occur in first a few periods and around the twelfthperiod. We might take considerations of seasonal ARIMA with frequency = 12.2

Noise: There is noise with time, so we should consider ARIMA model with noise.

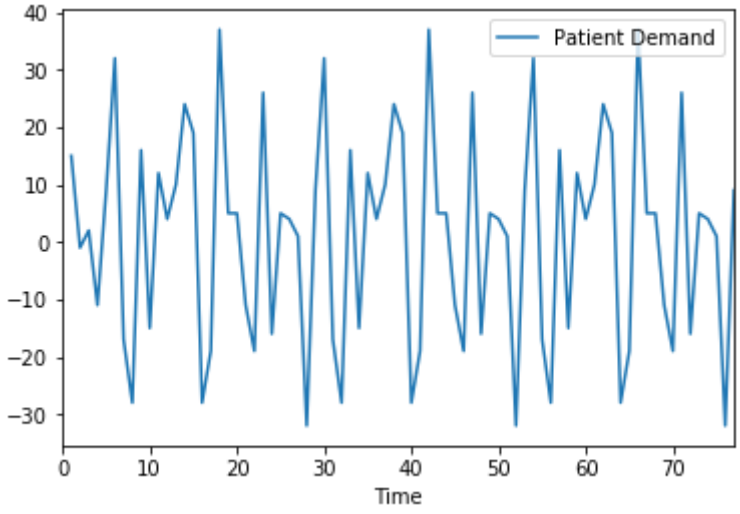
(2)

In [64]:
```
df_diff = df.diff()
df_diff.plot()
df_diff
```

Out[64]:

Time	Patient Demand
0	NaN
1	15.0
2	-1.0
3	2.0
4	-11.0
...	...
73	5.0
74	4.0
75	1.0
76	-32.0
77	9.0

78 rows × 1 columns



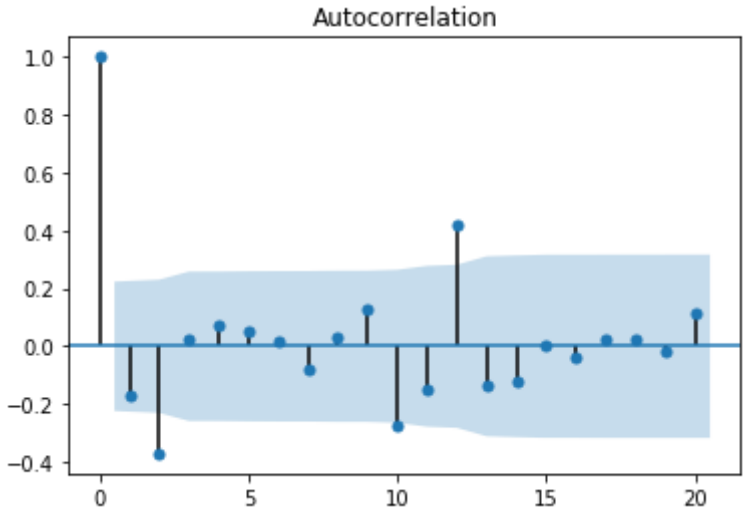
In [66]:
```
df_diff = df_diff.dropna()
df_diff
```

Out[66]:

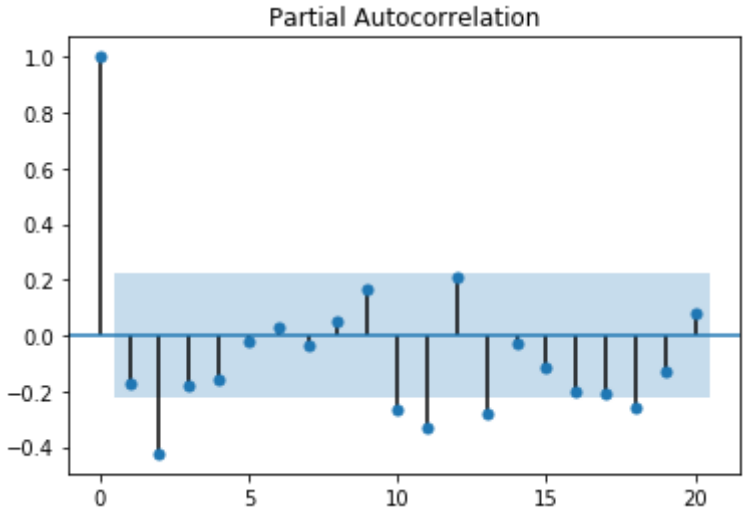
Time	Patient Demand
1	15.0
2	-1.0
3	2.0
4	-11.0
5	9.0
...	...
73	5.0
74	4.0
75	1.0
76	-32.0
77	9.0

77 rows × 1 columns

In [67]:
```
sm.graphics.tsa.plot_acf(df_diff.values.squeeze(), lags=20)
plt.show()
```



In [68]:
```
sm.graphics.tsa.plot_pacf(df_diff.values.squeeze(), lags=20)
plt.show()
```



We find out that time series plot after the first order differenciation are more stationary in range (-30,30)across 0. ACF and PACF both decay with time with the second lag cut, but big lags occur at time = 12.

In [103]:
```
#for non-seasonal
auto_arima(df, seasonal = False, m=12, error_action = "ignore").summary
()
```

```
/Users/yilinyin/opt/anaconda3/lib/python3.7/site-packages/pmdarima/arim
a/_validation.py:62: UserWarning: m (12) set for non-seasonal fit. Sett
ing to 0
  warnings.warn("m (%i) set for non-seasonal fit. Setting to 0" % m)
```

Out[103]:

Statespace Model Results

Dep. Variable:	y	**No. Observations:**	78
Model:	SARIMAX(2, 1, 1)	**Log Likelihood**	-325.520
Date:	Thu, 11 Feb 2021	**AIC**	659.039
Time:	23:53:49	**BIC**	668.414
Sample:	0	**HQIC**	662.789
	- 78		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.1006	0.233	0.431	0.666	-0.356	0.558
ar.L2	-0.3577	0.124	-2.884	0.004	-0.601	-0.115
ma.L1	-0.4229	0.270	-1.567	0.117	-0.952	0.106
sigma2	273.2425	69.239	3.946	0.000	137.537	408.948

Ljung-Box (Q):	94.18	**Jarque-Bera (JB):**	1.48
Prob(Q):	0.00	**Prob(JB):**	0.48
Heteroskedasticity (H):	1.15	**Skew:**	-0.02
Prob(H) (two-sided):	0.73	**Kurtosis:**	2.32

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

ARIMA(2,1,1) is the best fit for non-seasonal model

In [153]:
```
#for seasonal
stepwise_fit = auto_arima(df, start_p = 0
                          , start_q = 0,max_p =5, max_q = 5
                          , m=12, seasonal = True, d=None,
                          trace = True,
                          error_action ='ignore'
                          , suppress_warnings = True,
                          stepwise = True)
```

```
Performing stepwise search to minimize aic
 ARIMA(0,0,0)(1,0,1)[12] intercept   : AIC=inf, Time=0.18 sec
 ARIMA(0,0,0)(0,0,0)[12] intercept   : AIC=673.823, Time=0.01 sec
 ARIMA(1,0,0)(1,0,0)[12] intercept   : AIC=660.441, Time=0.11 sec
 ARIMA(0,0,1)(0,0,1)[12] intercept   : AIC=665.036, Time=0.08 sec
 ARIMA(0,0,0)(0,0,0)[12]             : AIC=672.096, Time=0.01 sec
 ARIMA(1,0,0)(0,0,0)[12] intercept   : AIC=673.492, Time=0.03 sec
 ARIMA(1,0,0)(2,0,0)[12] intercept   : AIC=inf, Time=0.31 sec
 ARIMA(1,0,0)(1,0,1)[12] intercept   : AIC=inf, Time=0.25 sec
 ARIMA(1,0,0)(0,0,1)[12] intercept   : AIC=669.699, Time=0.07 sec
 ARIMA(1,0,0)(2,0,1)[12] intercept   : AIC=493.246, Time=0.34 sec
 ARIMA(1,0,0)(2,0,2)[12] intercept   : AIC=495.235, Time=0.48 sec
 ARIMA(1,0,0)(1,0,2)[12] intercept   : AIC=inf, Time=0.49 sec
 ARIMA(0,0,0)(2,0,1)[12] intercept   : AIC=491.755, Time=0.22 sec
 ARIMA(0,0,0)(2,0,0)[12] intercept   : AIC=inf, Time=0.18 sec
 ARIMA(0,0,0)(2,0,2)[12] intercept   : AIC=493.744, Time=0.38 sec
 ARIMA(0,0,0)(1,0,0)[12] intercept   : AIC=658.839, Time=0.05 sec
 ARIMA(0,0,0)(1,0,2)[12] intercept   : AIC=inf, Time=0.38 sec
 ARIMA(0,0,1)(2,0,1)[12] intercept   : AIC=492.836, Time=0.32 sec
 ARIMA(1,0,1)(2,0,1)[12] intercept   : AIC=492.113, Time=0.49 sec
 ARIMA(0,0,0)(2,0,1)[12]             : AIC=489.874, Time=0.30 sec
 ARIMA(0,0,0)(1,0,1)[12]             : AIC=inf, Time=0.10 sec
 ARIMA(0,0,0)(2,0,0)[12]             : AIC=inf, Time=0.07 sec
 ARIMA(0,0,0)(2,0,2)[12]             : AIC=492.207, Time=0.49 sec
 ARIMA(0,0,0)(1,0,0)[12]             : AIC=656.978, Time=0.02 sec
 ARIMA(0,0,0)(1,0,2)[12]             : AIC=inf, Time=0.27 sec
 ARIMA(1,0,0)(2,0,1)[12]             : AIC=491.389, Time=0.26 sec
 ARIMA(0,0,1)(2,0,1)[12]             : AIC=491.003, Time=0.25 sec
 ARIMA(1,0,1)(2,0,1)[12]             : AIC=492.570, Time=0.41 sec

Best model:  ARIMA(0,0,0)(2,0,1)[12]
Total fit time: 6.566 seconds
```

auto_arima suggests that the best model is SARIMA model (0,1,0) with seasonal order (2,0,1,12). SARIMA stands for Seasonal AutoRegressive Integrated Moving Average

In [152]:
```
fitted_SARIMA_model = SARIMAX(df, order =(0, 1, 0), seasonal_order = (2,
0,1,12))
s_results = fitted_SARIMA_model.fit()
s_results.summary()
#error
```

```
/Users/yilinyin/opt/anaconda3/lib/python3.7/site-packages/statsmodels/t
sa/statespace/sarimax.py:981: UserWarning: Non-stationary starting seas
onal autoregressive Using zeros as starting parameters.
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
statsmodels/tsa/statespace/_filters/_inversions.pyx in statsmodels.tsa.statespace._filters._inversions.dinverse_univariate()

ZeroDivisionError: float division

During handling of the above exception, another exception occurred:

LinAlgError                               Traceback (most recent call last)
<ipython-input-152-7408844ecaa7> in <module>
      1 fitted_SARIMA_model = SARIMAX(df, order =(0, 1, 0), seasonal_order = (2,0,1,12))
----> 2 s_results = fitted_SARIMA_model.fit()
      3 s_results.summary()

~/opt/anaconda3/lib/python3.7/site-packages/statsmodels/tsa/statespace/mlemodel.py in fit(self, start_params, transformed, cov_type, cov_kwds, method, maxiter, full_output, disp, callback, return_params, optim_score, optim_complex_step, optim_hessian, flags, **kwargs)
    480         k_params = len(self.param_names)
    481         # Initialization (this is done here rather than in the constructor
--> 482         # because param_names may not be available at that point)
    483         if self._fixed_params is None:
    484             self._fixed_params = {}

~/opt/anaconda3/lib/python3.7/site-packages/statsmodels/base/model.py in fit(self, start_params, method, maxiter, full_output, disp, fargs, callback, retall, skip_hessian, **kwargs)
    468                     tolerance for termination. Other arguments are mapped from
    469                     explicit argument of `fit`:
--> 470                       - `args` <- `fargs`
    471                       - `jac` <- `score`
    472                       - `hess` <- `hess`

~/opt/anaconda3/lib/python3.7/site-packages/statsmodels/base/optimizer.py in _fit(self, objective, gradient, start_params, fargs, kwargs, hessian, method, maxiter, full_output, disp, callback, retall)
    217
    218         #NOTE: fit_regularized checks the methods for these but it should be
--> 219         #      moved up probably
    220         if extra_fit_funcs:
    221             fit_funcs.update(extra_fit_funcs)

~/opt/anaconda3/lib/python3.7/site-packages/statsmodels/base/optimizer.py in _fit_lbfgs(f, score, start_params, fargs, kwargs, disp, maxiter, callback, retall, full_output, hess)
    437         warnflag = 0
    438         if disp:
--> 439             print("Optimization terminated successfully.")
```

```
    440             print("         Current function value: %f" % fval)
    441             print("         Iterations %d" % iterations)

~/opt/anaconda3/lib/python3.7/site-packages/scipy/optimize/lbfgsb.py in
fmin_l_bfgs_b(func, x0, fprime, args, approx_grad, bounds, m, factr, pg
tol, epsilon, iprint, maxfun, maxiter, disp, callback, maxls)
    197     res = _minimize_lbfgsb(fun, x0, args=args, jac=jac, bounds=
bounds,
    198                            **opts)
--> 199     d = {'grad': res['jac'],
    200          'task': res['message'],
    201          'funcalls': res['nfev'],

~/opt/anaconda3/lib/python3.7/site-packages/scipy/optimize/lbfgsb.py in
_minimize_lbfgsb(fun, x0, args, jac, bounds, disp, maxcor, ftol, gtol,
 eps, maxfun, maxiter, iprint, callback, maxls, **unknown_options)
    333
    334     x = array(x0, float64)
--> 335     f = array(0.0, float64)
    336     g = zeros((n,), float64)
    337     wa = zeros(2*m*n + 5*n + 11*m*m + 8*m, float64)

~/opt/anaconda3/lib/python3.7/site-packages/scipy/optimize/lbfgsb.py in
func_and_grad(x)
    279     n, = x0.shape
    280
--> 281     if bounds is None:
    282         bounds = [(None, None)] * n
    283     if len(bounds) != n:

~/opt/anaconda3/lib/python3.7/site-packages/scipy/optimize/optimize.py
 in _approx_fprime_helper(xk, f, epsilon, args, f0)
    694             raise ValueError("`initial_simplex` should be an ar
ray of shape (N+1,N)")
    695         if len(x0) != sim.shape[1]:
--> 696             raise ValueError("Size of `initial_simplex` is not
 consistent with `x0`")
    697         N = sim.shape[1]
    698

~/opt/anaconda3/lib/python3.7/site-packages/scipy/optimize/optimize.py
 in function_wrapper(*wrapper_args)
    324     >>> from mpl_toolkits.mplot3d import Axes3D
    325     >>> x = np.linspace(-1, 1, 50)
--> 326     >>> X, Y = np.meshgrid(x, x)
    327     >>> ax = plt.subplot(111, projection='3d')
    328     >>> ax.plot_surface(X, Y, rosen([X, Y]))

~/opt/anaconda3/lib/python3.7/site-packages/statsmodels/base/model.py i
n f(params, *args)
    442                     Relative error in loglike(params) for accep
table for
    443                     convergence.
--> 444                 maxfun : int
    445                     Maximum number of function evaluations to m
ake.
    446                 start_direc : ndarray
```

```
~/opt/anaconda3/lib/python3.7/site-packages/statsmodels/tsa/statespace/
mlemodel.py in loglike(self, params, *args, **kwargs)
    657         elif optim_complex_step and self.ssm._complex_endog:
    658             raise ValueError('Cannot use complex step derivativ
es when data'
--> 659                              ' or parameters are complex.')
    660
    661         # Standardize starting parameters

~/opt/anaconda3/lib/python3.7/site-packages/statsmodels/tsa/statespace/
kalman_filter.py in loglike(self, **kwargs)
    873             self.filter_concentrated = False
    874             self._scale = scale
--> 875             obs_cov = self['obs_cov']
    876             state_cov = self['state_cov']
    877             self['obs_cov'] = scale * obs_cov

~/opt/anaconda3/lib/python3.7/site-packages/statsmodels/tsa/statespace/
kalman_filter.py in _filter(self, filter_method, inversion_method, stab
ility_method, conserve_memory, filter_timing, tolerance, loglikelihood_
burn, complex_step)
    800
    801         Examples
--> 802         --------
    803         >>> mod = sm.tsa.statespace.SARIMAX(range(10))
    804         >>> mod.ssm..conserve_memory

statsmodels/tsa/statespace/_kalman_filter.pyx in statsmodels.tsa.states
pace._kalman_filter.dKalmanFilter.__call__()

statsmodels/tsa/statespace/_kalman_filter.pyx in statsmodels.tsa.states
pace._kalman_filter.dKalmanFilter.__next__()

statsmodels/tsa/statespace/_filters/_inversions.pyx in statsmodels.tsa.
statespace._filters._inversions.dinverse_univariate()

LinAlgError: Non-positive-definite forecast error covariance matrix enc
ountered at period 1
```

In [148]:
```
start = 1
end = len(df)-1
SARIMA_predictions = s_results.predict(start = start, end = end).rename(
'SARIMA Predictions')
np.mean(meanabs(df, SARIMA_predictions))
SARIMA_predictions
```

Out[148]:
```
Time
1       0.397774
2       0.697339
3       0.921392
4       1.089228
5       1.205654
          ...
73    605.340023
74    631.349802
75    628.576233
76    589.890986
77    568.813166
Name: SARIMA Predictions, Length: 77, dtype: float64
```

In [139]:
```
#non-seasonal model
model = ARIMA(df, order=(2, 1, 1))
results = model.fit()
results.summary()
```

Out[139]:

ARIMA Model Results

Dep. Variable:	D.Patient Demand	**No. Observations:**	77
Model:	ARIMA(2, 1, 1)	**Log Likelihood**	-324.574
Method:	css-mle	**S.D. of innovations**	16.321
Date:	Fri, 12 Feb 2021	**AIC**	659.148
Time:	00:22:40	**BIC**	670.867
Sample:	1	**HQIC**	663.835

	coef	std err	z	P>|z|	[0.025	0.975]
const	1.1540	0.792	1.457	0.149	-0.398	2.706
ar.L1.D.Patient Demand	0.1460	0.194	0.752	0.455	-0.235	0.527
ar.L2.D.Patient Demand	-0.3537	0.126	-2.814	0.006	-0.600	-0.107
ma.L1.D.Patient Demand	-0.4958	0.200	-2.484	0.015	-0.887	-0.105

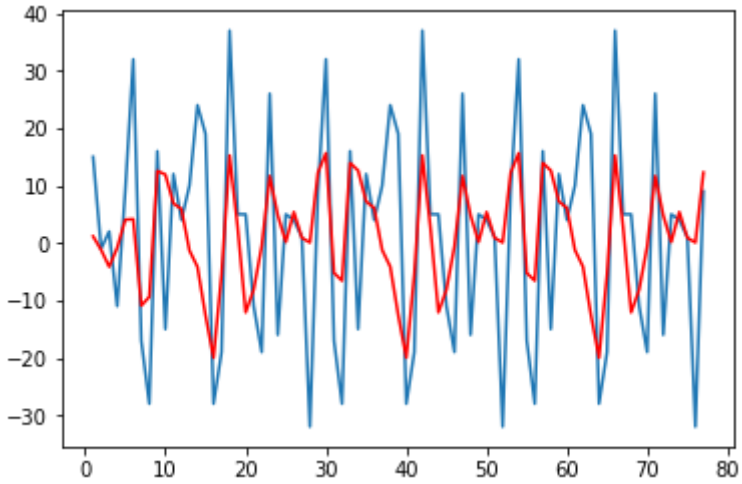
Roots

	Real	Imaginary	Modulus	Frequency
AR.1	0.2064	-1.6687j	1.6814	-0.2304
AR.2	0.2064	+1.6687j	1.6814	0.2304
MA.1	2.0169	+0.0000j	2.0169	0.0000

In [140]:
```
start = 1
end = len(df_diff)
ARIMA_predictions = results.predict(start = start, end = end).rename('AR
IMA Predictions')
np.mean(meanabs(df_diff, ARIMA_predictions))
```

Out[140]:
```
17.137847473758985
```

In [141]:
```
plt.plot(df_diff)
plt.plot(ARIMA_predictions, color='red')
plt.show()
```



The performance of non-seasonal ARIMA has large abs error.