In [4]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

In [5]:
```
df = pd.read_csv('zinc_prices_IMF.csv')
df.head()
```

Out[5]:

	Date	Price
0	1-Jan-80	773.82
1	1-Feb-80	868.62
2	1-Mar-80	740.75
3	1-Apr-80	707.68
4	1-May-80	701.07

In [6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 434 entries, 0 to 433
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Date    434 non-null    object 
 1   Price   434 non-null    float64
dtypes: float64(1), object(1)
memory usage: 6.9+ KB
```

In [7]:
```
# convertinng date object  to datetime format
df['Date'] = pd.to_datetime(df['Date'])
```

In [8]:
```
df.head()
```

Out[8]:

	Date	Price
0	1980-01-01	773.82
1	1980-02-01	868.62
2	1980-03-01	740.75
3	1980-04-01	707.68
4	1980-05-01	701.07

In [9]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 434 entries, 0 to 433
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Date    434 non-null    datetime64[ns]
 1   Price   434 non-null    float64
dtypes: datetime64[ns](1), float64(1)
memory usage: 6.9 KB
```

In [10]:
```
## Date colun is no converetd to Datetime format
```

In [11]:
```
df.set_index('Date', inplace=True)
```

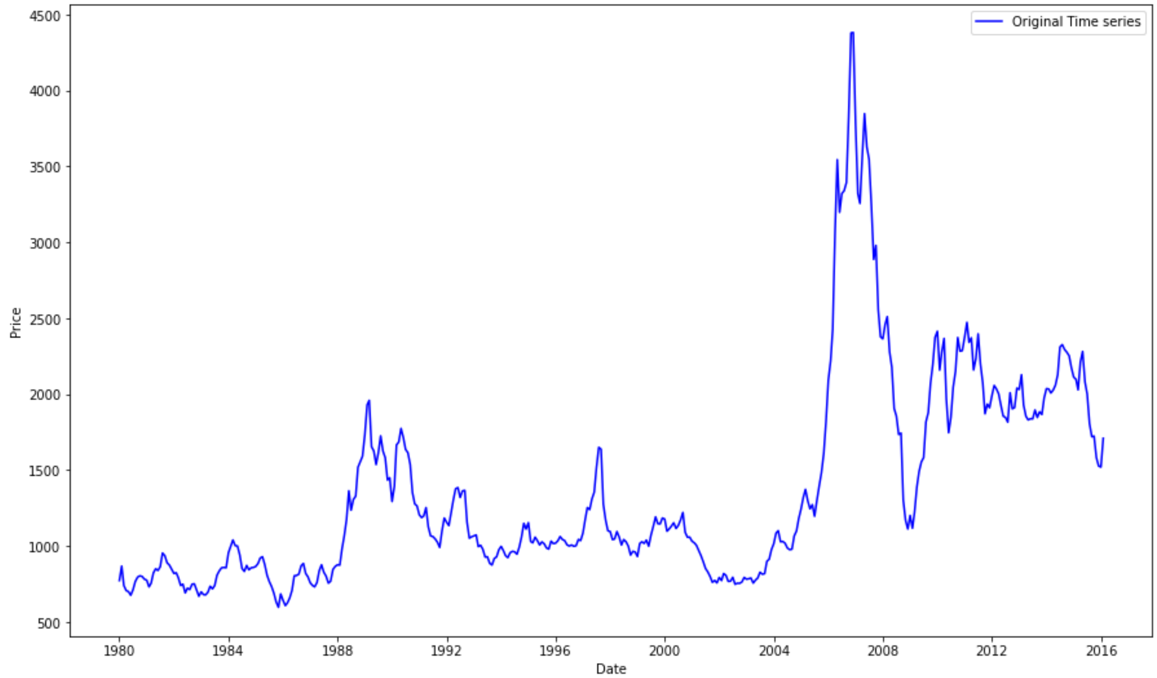
In [12]:
```
df.head()
```

Out[12]:

Date	Price
1980-01-01	773.82
1980-02-01	868.62
1980-03-01	740.75
1980-04-01	707.68
1980-05-01	701.07

Visualize the time series

In [13]:
```
plt.figure(figsize=(15,9))
plt.plot(df['Price'], color='blue', label='Original Time series')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend(loc='best')
plt.show()
```



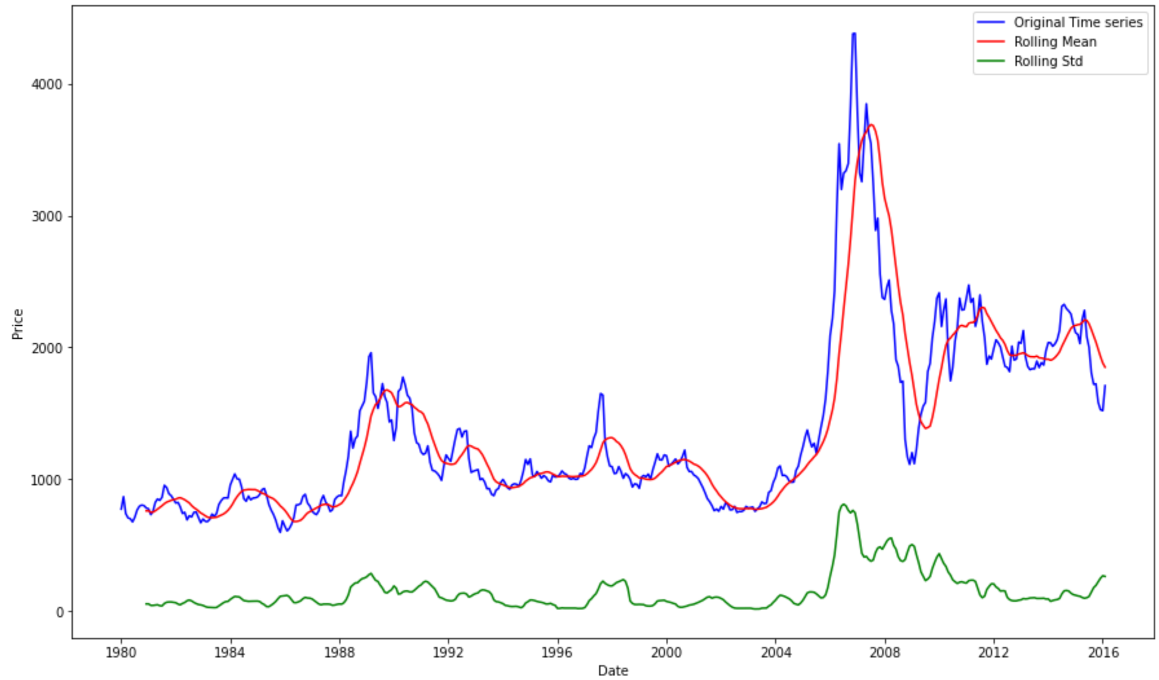
Check for the stationarity of your data using Rolling Statistics and Dickey-Fuller test and if present,

remove it using the stationarity removal techniques

In [14]:
```
# Checking Stationarity using Rolling Statistics

ts_rolling_mean = df['Price'].rolling(12).mean()
ts_rolling_std = df['Price'].rolling(12).std()
```

In [15]:
```
plt.figure(figsize=(15,9))
plt.plot(df['Price'], color='blue', label='Original Time series')
plt.plot(ts_rolling_mean, color='red', label='Rolling Mean')
plt.plot(ts_rolling_std, color='green', label='Rolling Std')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend(loc='best')
plt.show()
```



Observations:

* by looking at the rolling statistics, # we see that the rolling mean is in upward trend and the rolling std is slightly increasing,therefore wecan say that the time series is Non - Stationary

In [16]:
```
#checking the non stationarity using Dickey-Fuller test

# Null hypo thesis  = Time series in Non Stationary


from statsmodels.tsa.stattools import adfuller
```

In [17]:
```
adf_test = adfuller(df['Price'])
adf_test
```

Out[17]:
```
(-3.139600554153096,
 0.023758021886101866,
 7,
 426,
 {'1%': -3.4457939940402107,
  '5%': -2.8683485906158963,
  '10%': -2.570396746236417},
 5069.9473477604715)
```

In [18]:
```
print(f'test statistics {adf_test[0]}')
print(f'P-value {adf_test[1]}')
print(f'Lags Used {adf_test[2]}')
print(f'No of observations {adf_test[3]}')
```

```
test statistics -3.139600554153096
P-value 0.023758021886101866
Lags Used 7
No of observations 426
```

Observations:

```
# p value is > 0.05, hence we accept the null hypothesis and say that the
time series is Non Stationary
```

Applying diferencing to make the time series stationary

In [19]:
```
nshifts=1
ts_zinc =df['Price']-df['Price'].shift(nshifts)
adfuller(ts_zinc[nshifts:])[1]
```

Out[19]:
```
6.19820746102993e-06
```

In [20]:
```
nshifts=2
ts_zinc =df['Price']-df['Price'].shift(nshifts)
adfuller(ts_zinc[nshifts:])[1]
```

Out[20]:
```
4.420793207125775e-06
```

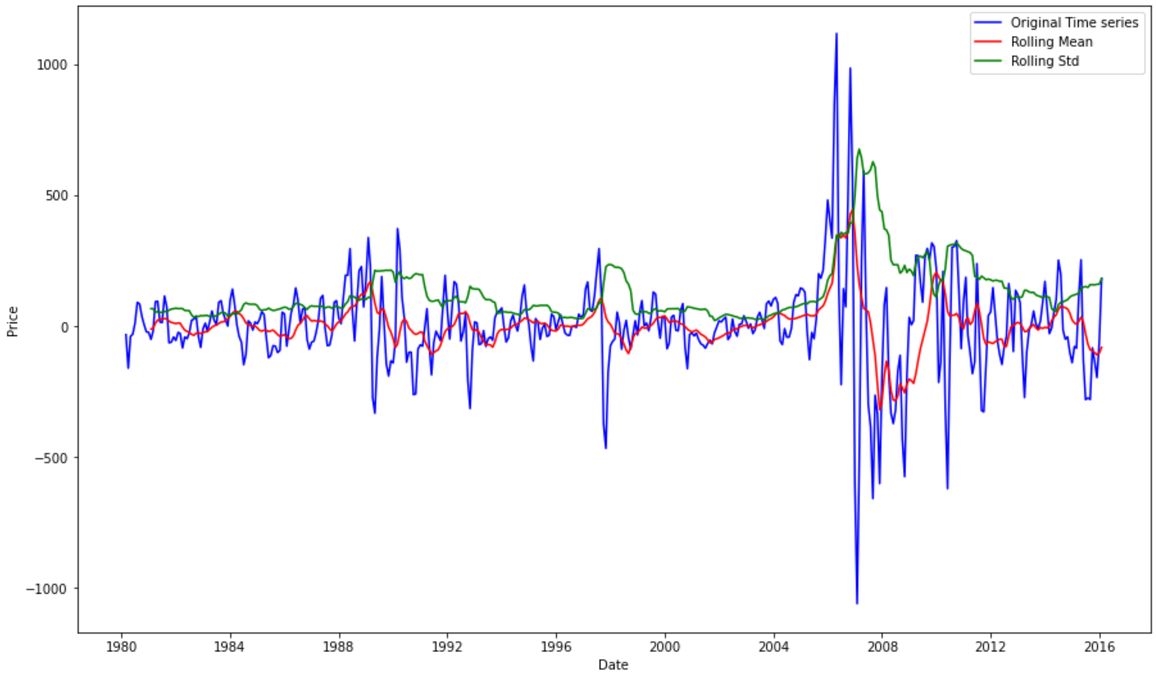
Observations:

```
# so while apply the 1st degree diferencing, we get that p value =6.198207
46102993e-06 = 0.00000619 which is < 0.005,
# hence we can reject the null hypothesis now and say that the time series
is now stationary
```

In [21]:
```
# Plotting the timeseries after making the time series stationary

ts_zinc_rolling_mean = ts_zinc.rolling(12).mean()
ts_zinc_rolling_std = ts_zinc.rolling(12).std()

plt.figure(figsize=(15,9))
plt.plot(ts_zinc, color='blue', label='Original Time series')
plt.plot(ts_zinc_rolling_mean, color='red', label='Rolling Mean')
plt.plot(ts_zinc_rolling_std, color='green', label='Rolling Std')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend(loc='best')
plt.show()
```



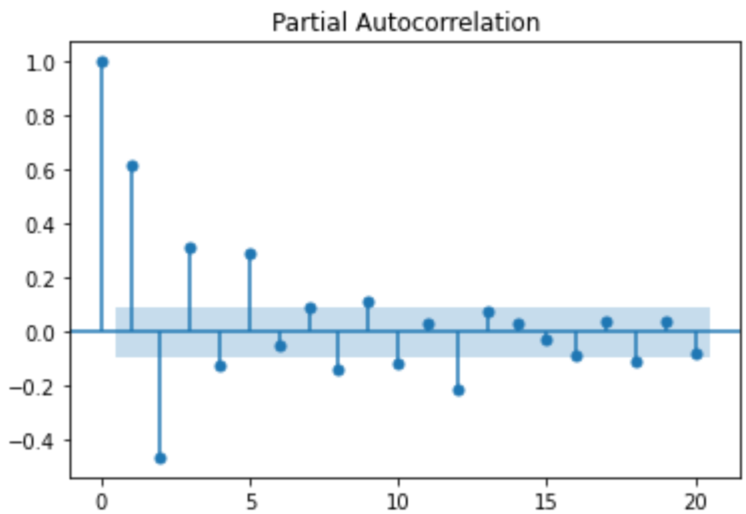
Hence the value for d = 1

In [ ]:

Plot ACF and PACF plots. Find the p and q values

In [22]:
```
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
```

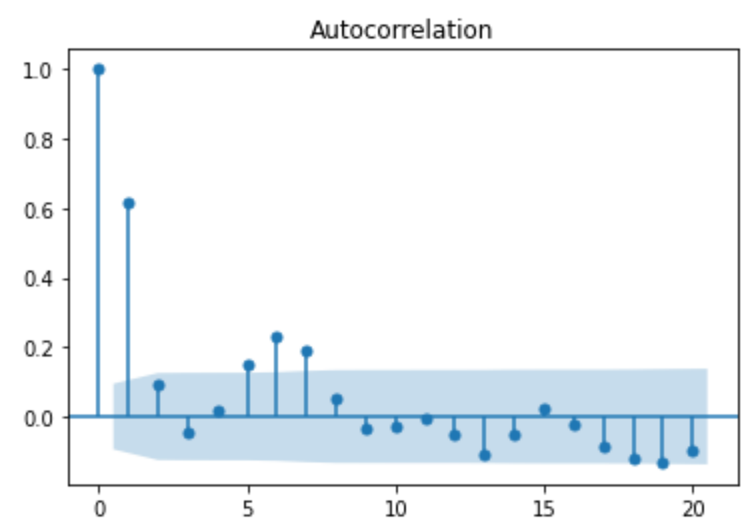
In [23]:
```
plot_pacf(ts_zinc[nshifts:],lags=20)
plt.show()
```



here the the pacf, goes below zero after after lag=2, or p order can be either 1 or 2

so, we can take p = 2

In [24]:
```
plot_acf(ts_zinc[nshifts:],lags=20)
plt.show()
```



here the acf, the point at which it has cross the upper confidence bound, so, q = 1

In [ ]:

Perform ARIMA modeling

In [25]:
```
df.shape[0]/2
```
Out[25]:
```
217.0
```

In [26]:
```
# Splitting the data into trai and test
train = df['Price'][:217]
test = df['Price'][217:]

print(train.shape)
print(test.shape)
```
```
(217,)
(217,)
```

In [ ]:

In [27]:
```
# Build Model
from statsmodels.tsa.arima_model import ARIMA
model = ARIMA(train, order=(2, 1, 1))
fitted = model.fit(disp=-1)
```
```
/usr/local/lib/python3.7/site-packages/statsmodels/tsa/base/tsa_mode
l.py:162: ValueWarning: No frequency information was provided, so inf
erred frequency MS will be used.
  % freq, ValueWarning)
/usr/local/lib/python3.7/site-packages/statsmodels/tsa/base/tsa_mode
l.py:162: ValueWarning: No frequency information was provided, so inf
erred frequency MS will be used.
  % freq, ValueWarning)
```

In [28]:
```
fitted.summary()
```

Out[28]:

ARIMA Model Results

Dep. Variable:	D.Price	**No. Observations:**	216
Model:	ARIMA(2, 1, 1)	**Log Likelihood**	-1218.797
Method:	css-mle	**S.D. of innovations**	67.943
Date:	Mon, 11 Oct 2021	**AIC**	2447.594
Time:	16:05:55	**BIC**	2464.470
Sample:	02-01-1980	**HQIC**	2454.412
	- 01-01-1998		

	coef	std err	z	P>|z|	[0.025	0.975]
const	2.0584	1.300	1.583	0.113	-0.490	4.607
ar.L1.D.Price	1.2234	0.065	18.689	0.000	1.095	1.352
ar.L2.D.Price	-0.2705	0.065	-4.130	0.000	-0.399	-0.142
ma.L1.D.Price	-0.9999	0.014	-73.932	0.000	-1.026	-0.973

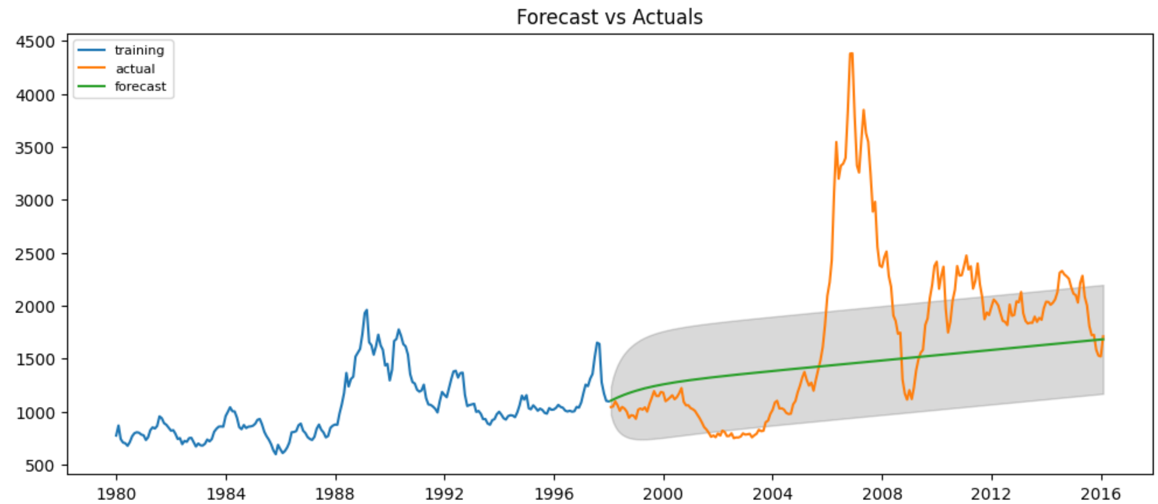
Roots

	Real	Imaginary	Modulus	Frequency
AR.1	1.0709	+0.0000j	1.0709	0.0000
AR.2	3.4526	+0.0000j	3.4526	0.0000
MA.1	1.0001	+0.0000j	1.0001	0.0000

In [29]:
```
# Forecast
fc, se, conf = fitted.forecast(217, alpha=0.05)  # 95% conf
```

In [30]:
```
# Make as pandas series
fc_series = pd.Series(fc, index=test.index)
lower_series = pd.Series(conf[:, 0], index=test.index)
upper_series = pd.Series(conf[:, 1], index=test.index)
```

In [31]:
```
# Plot
plt.figure(figsize=(12,5), dpi=100)
plt.plot(train, label='training')
plt.plot(test, label='actual')
plt.plot(fc_series, label='forecast')
plt.fill_between(lower_series.index, lower_series, upper_series,
                 color='k', alpha=.15)
plt.title('Forecast vs Actuals')
plt.legend(loc='upper left', fontsize=8)
plt.show()
```



In [32]:
```
from sklearn.metrics import mean_absolute_percentage_error
```

In [33]:
```
1-mean_absolute_percentage_error(fc_series,test.values)
```

Out[33]: 0.6351159647009385

With p =2, d= 1 and q = 1, we see that the p value is < 0.05 but the mean_absolute_percentage_error is just around 63.5%

In [34]:
```
train = df['Price'][:400]
test = df['Price'][400:]

print(train.shape)
print(test.shape)

# Build Model
from statsmodels.tsa.arima_model import ARIMA
model = ARIMA(train, order=(1, 2, 1))
fitted = model.fit(disp=-1)
fitted.summary()
```

```
(400,)
(34,)

/usr/local/lib/python3.7/site-packages/statsmodels/tsa/base/tsa_mode
l.py:162: ValueWarning: No frequency information was provided, so inf
erred frequency MS will be used.
  % freq, ValueWarning)
/usr/local/lib/python3.7/site-packages/statsmodels/tsa/base/tsa_mode
l.py:162: ValueWarning: No frequency information was provided, so inf
erred frequency MS will be used.
  % freq, ValueWarning)
```

Out[34]:

ARIMA Model Results

Dep. Variable:	D2.Price	**No. Observations:**	398
Model:	ARIMA(1, 2, 1)	**Log Likelihood**	-2439.441
Method:	css-mle	**S.D. of innovations**	110.346
Date:	Mon, 11 Oct 2021	**AIC**	4886.882
Time:	16:06:00	**BIC**	4902.828
Sample:	03-01-1980	**HQIC**	4893.198
	- 04-01-2013		

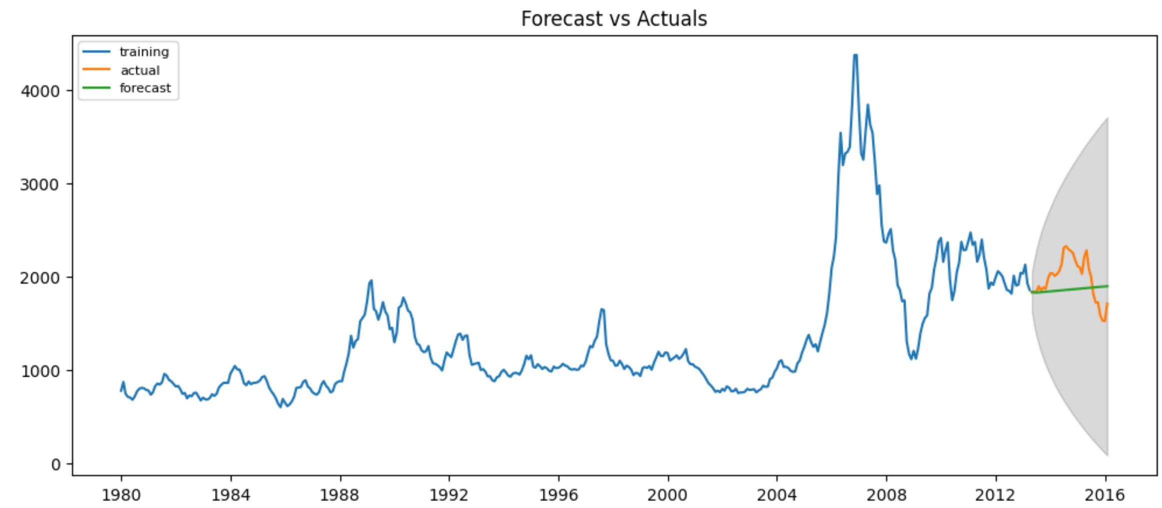
	coef	std err	z	P>|z|	[0.025	0.975]
const	-0.0026	0.070	-0.037	0.971	-0.139	0.134
ar.L1.D2.Price	0.3133	0.048	6.560	0.000	0.220	0.407
ma.L1.D2.Price	-0.9999	0.007	-147.682	0.000	-1.013	-0.987

Roots

	Real	Imaginary	Modulus	Frequency
AR.1	3.1918	+0.0000j	3.1918	0.0000
MA.1	1.0001	+0.0000j	1.0001	0.0000

In [35]:
```
# Forecast
fc, se, conf = fitted.forecast(34, alpha=0.05)  # 95% conf
# Make as pandas series
fc_series = pd.Series(fc, index=test.index)
lower_series = pd.Series(conf[:, 0], index=test.index)
upper_series = pd.Series(conf[:, 1], index=test.index)

# Plot
plt.figure(figsize=(12,5), dpi=100)
plt.plot(train, label='training')
plt.plot(test, label='actual')
plt.plot(fc_series, label='forecast')
plt.fill_between(lower_series.index, lower_series, upper_series,
                 color='k', alpha=.15)
plt.title('Forecast vs Actuals')
plt.legend(loc='upper left', fontsize=8)
plt.show()
```



In [36]:
```
1-mean_absolute_percentage_error(fc_series,test.values)
```

Out[36]: 0.8843913316612979

reducing the test dataset to just 34 rows and with p =1, d= 1 and q = 1, we see that the p value is < 0.05 and the mean_absolute_percentage_error is just around 88.4%

In [ ]: