```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.arima_model import ARIMA
import tensorflow as tf
from statsmodels.tsa.arima.model import ARIMA
import warnings
warnings.filterwarnings("ignore")
from statsmodels.tsa.arima_model import ARIMA
from statsmodels.tsa.arima.model import ARIMA
```

```
import matplotlib.pyplot as plt
import pandas as pd
import statsmodels.api as sm
from statsmodels.graphics.tsaplots import plot_predict
from statsmodels.tsa.arima.model import ARIMA
```

```
df = pd.read_csv("/content/drive/MyDrive/TM/a10.csv",index_col = 'date')
```

```
df
```

	value
date	
1991-07-01	3.526591
1991-08-01	3.180891
1991-09-01	3.252221
1991-10-01	3.611003
1991-11-01	3.565869
...	...
2008-02-01	21.654285
2008-03-01	18.264945
2008-04-01	23.107677
2008-05-01	22.912510
2008-06-01	19.431740

204 rows × 1 columns

EDA

```
df.shape
```

```
(204, 1)
```

```
df=df.dropna()
```

```
df.isnull().sum()
```

```
value    0
dtype: int64
```

```
df.head(5)
```

	value
date	

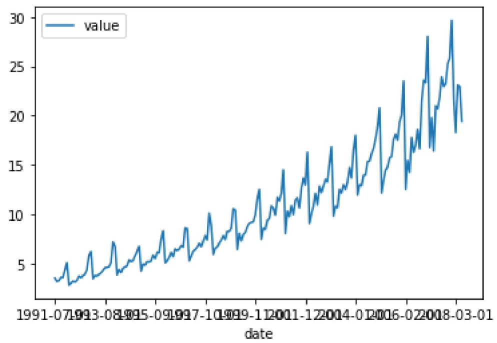
```
df.tail()
```

	value
date	
2008-02-01	21.654285
2008-03-01	18.264945
2008-04-01	23.107677
2008-05-01	22.912510
2008-06-01	19.431740

Visualize of Data

```
df.plot()
# seasonlity data
```

```
<AxesSubplot:xlabel='date'>
```




```
# test for stationarity
from statsmodels.tsa.stattools import adfuller
```

```
# check stationary or not
def adfuller_test(value):
  result = adfuller(value)
  labels = ['ADF Test Statistics','p-value','Number of observation used']
  for value,label in zip(result,labels):
    print(label+' : '+str(value))
  if result[1]<=0.05:
    print("strong evidence against the null hypothesis, reject the null hypothesis")
  else:
    print("weak evidence against the null hypothesis")
```

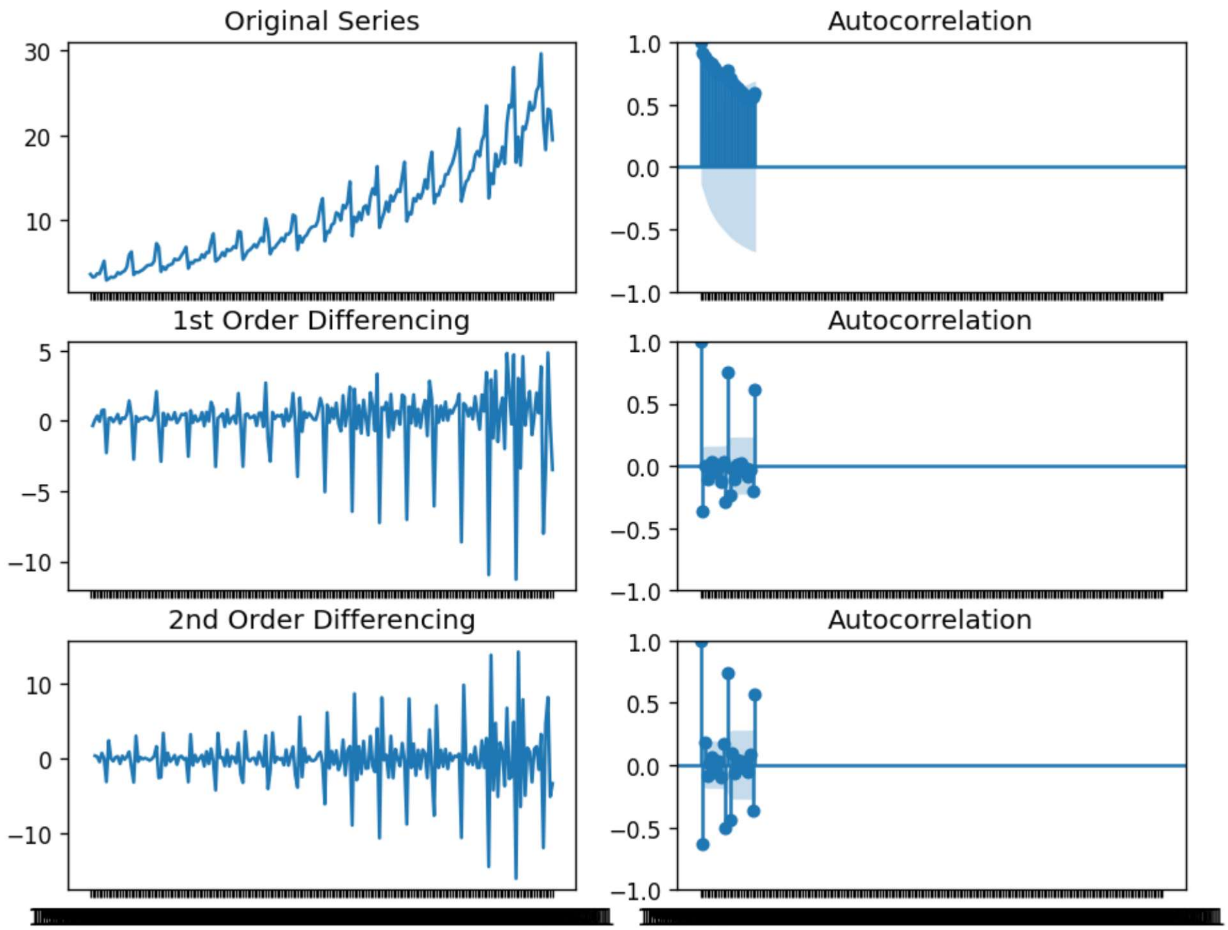
```
adfuller_test(df['value'])
```

```
ADF Test Statistics : 3.14518568930673
p-value : 1.0
Number of observation used : 15
weak evidence against the null hypothesis
```

Differencing

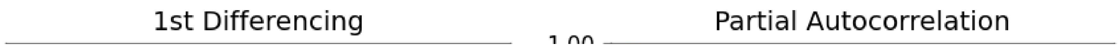
```
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

plt.rcParams.update({'figure.figsize':(9,7), 'figure.dpi':120})

# Original Series
fig, axes = plt.subplots(3, 2, sharex=True)
axes[0, 0].plot(df.value); axes[0, 0].set_title('Original Series')
plot_acf(df.value, ax=axes[0, 1])
```

```
# 1st Differencing
axes[1, 0].plot(df.value.diff()); axes[1, 0].set_title('1st Order Differencing')
plot_acf(df.value.diff().dropna(), ax=axes[1, 1])

# 2nd Differencing
axes[2, 0].plot(df.value.diff().diff()); axes[2, 0].set_title('2nd Order Differencing')
plot_acf(df.value.diff().diff().dropna(), ax=axes[2, 1])

plt.show()
```

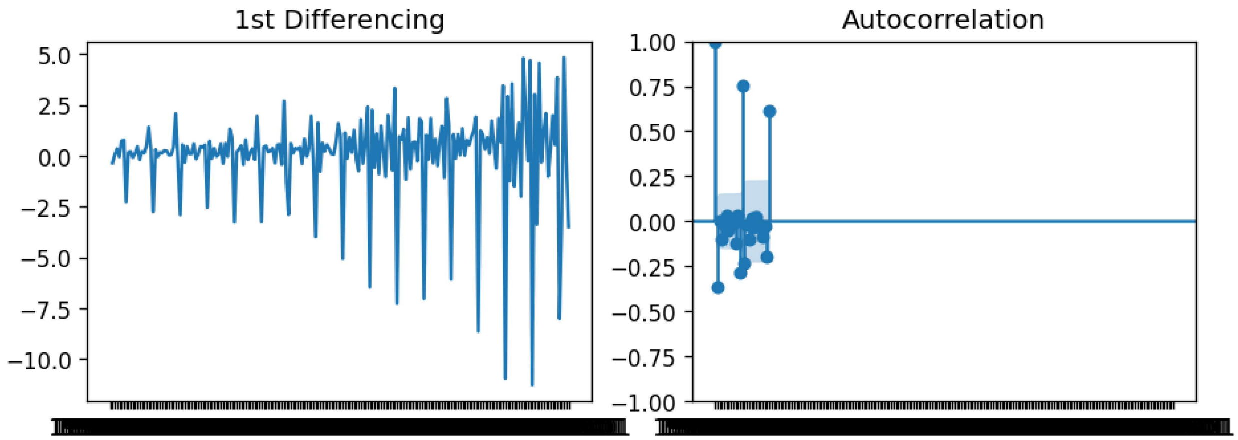



```
# PACF plot of 1st differenced series
plt.rcParams.update({'figure.figsize':(9,3), 'figure.dpi':120})

fig, axes = plt.subplots(1, 2, sharex=True)
axes[0].plot(df.value.diff()); axes[0].set_title('1st Differencing')
axes[1].set(ylim=(0,5))
plot_pacf(df.value.diff().dropna(), ax=axes[1])

plt.show()
```

```
/usr/local/lib/python3.9/dist-packages/statsmodels/graphics/tsaplots.py:348: FutureWarning: The default method 'yw' can pro
  warnings.warn(
```




```
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
plt.rcParams.update({'figure.figsize':(9,3), 'figure.dpi':120})

fig, axes = plt.subplots(1, 2, sharex=True)
axes[0].plot(df.value.diff()); axes[0].set_title('1st Differencing')
axes[1].set(ylim=(0,1.2))
plot_acf(df.value.diff().dropna(), ax=axes[1])

plt.show()
```




```
import statsmodels.api as sm
```

```
import statsmodels.api as smapi

# 1,1,2 ARIMA Model
model = ARIMA(df.value, order=(1,1,2))
model_fit = model.fit()
print(model_fit.summary())
```

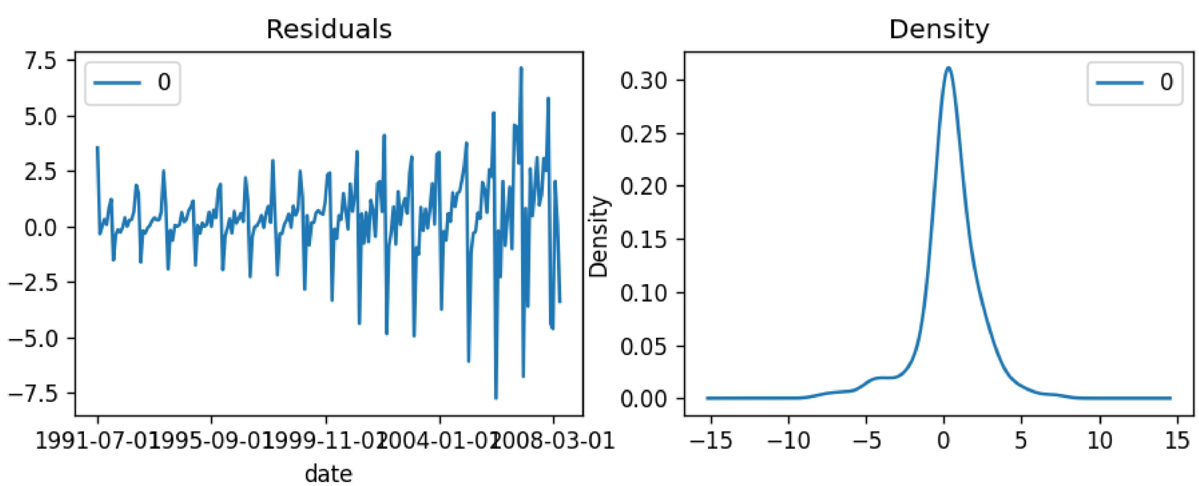
```
                               SARIMAX Results
==============================================================================
Dep. Variable:                  value   No. Observations:                  204
Model:                 ARIMA(1, 1, 2)   Log Likelihood                -424.570
Date:                Wed, 15 Mar 2023   AIC                            857.140
Time:                        08:49:23   BIC                            870.393
Sample:                    07-01-1991   HQIC                           862.502
                         - 06-01-2008
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.4178      0.356      1.174      0.240      -0.280       1.115
ma.L1         -0.9546      0.377     -2.531      0.011      -1.694      -0.215
ma.L2          0.0969      0.272      0.356      0.722      -0.437       0.631
sigma2         3.8259      0.269     14.209      0.000       3.298       4.354
===================================================================================
Ljung-Box (L1) (Q):                   0.46   Jarque-Bera (JB):               135.61
Prob(Q):                              0.50   Prob(JB):                         0.00
Heteroskedasticity (H):               9.82   Skew:                            -0.80
Prob(H) (two-sided):                  0.00   Kurtosis:                         6.67
===================================================================================

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

```
# 1,1,1 ARIMA Model
model = ARIMA(df.value, order=(1,1,1))
model_fit = model.fit()
print(model_fit.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                  value   No. Observations:                  204
```

```
Model:                 ARIMA(1, 1, 1)   Log Likelihood                -424.762
Date:                Wed, 15 Mar 2023   AIC                            855.524
Time:                        08:50:00   BIC                            865.463
Sample:                    07-01-1991   HQIC                           859.545
                         - 06-01-2008
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.3009      0.094      3.195      0.001       0.116       0.485
ma.L1         -0.8300      0.048    -17.204      0.000      -0.925      -0.735
sigma2         3.8327      0.259     14.790      0.000       3.325       4.341
===================================================================================
Ljung-Box (L1) (Q):                   0.72   Jarque-Bera (JB):               130.26
Prob(Q):                              0.40   Prob(JB):                         0.00
Heteroskedasticity (H):               9.98   Skew:                            -0.75
Prob(H) (two-sided):                  0.00   Kurtosis:                         6.63
===================================================================================

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

```
# Plot residual errors
residuals = pd.DataFrame(model_fit.resid)
fig, ax = plt.subplots(1,2)
residuals.plot(title="Residuals", ax=ax[0])
residuals.plot(kind='kde', title='Density', ax=ax[1])
plt.show()
```



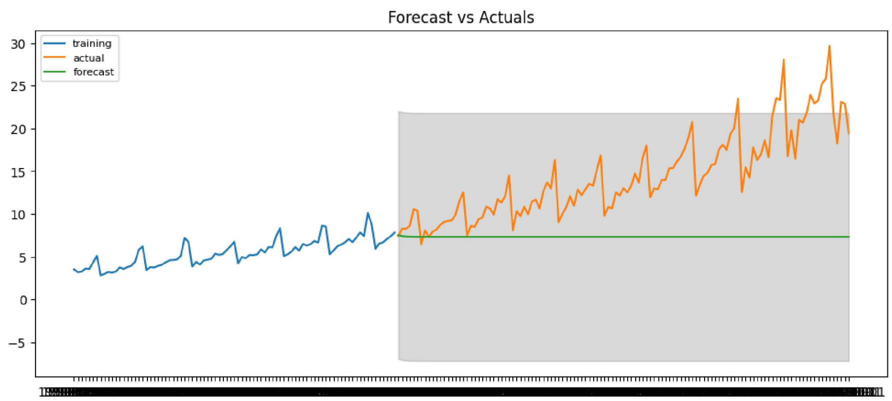

```
# Actual vs Fitted
model_fit.plot_predict()
plt.show()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-77-1705293967f2> in <module>
      1 # Actual vs Fitted
----> 2 model_fit.plot_predict()
      3 plt.show()

/usr/local/lib/python3.9/dist-packages/statsmodels/base/wrapper.py in __getattribute__(self, attr)
     32             pass
     33 
---> 34         obj = getattr(results, attr)
     35         data = results.model.data
     36         how = self._wrap_attrs.get(attr)

AttributeError: 'ARIMAResults' object has no attribute 'plot_predict'
```

SEARCH STACK OVERFLOW

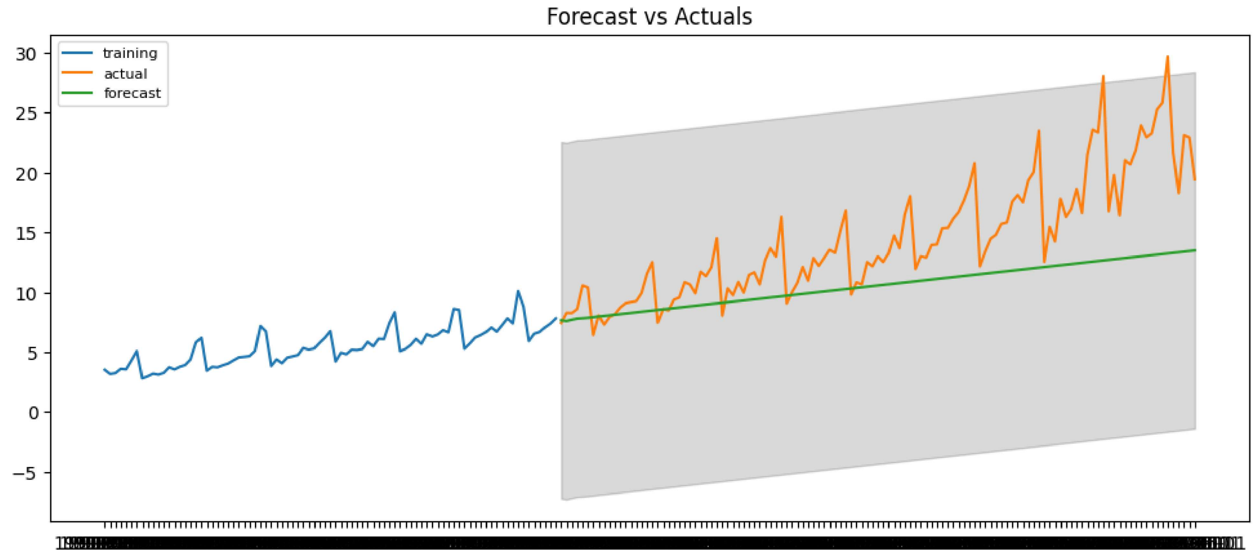
```
from statsmodels.tsa.stattools import acf

# Create Training and Test
train = df.value[:85]
test = df.value[85:]
```

```
# Build Model
# model = ARIMA(train, order=(3,2,1))
model = ARIMA(train, order=(1, 1, 1))
fitted = model.fit()

# Forecast
fc = fitted.forecast(119, alpha=0.05)
fc_series = pd.Series(fc, index=test.index)
se = fitted.forecast(119, alpha=0.05)[1]
lower_series = pd.Series(fc - 1.96*se, index=test.index)
upper_series = pd.Series(fc + 1.96*se, 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()
```




```
# Build Model
model = ARIMA(train, order=(3, 2, 1))
fitted = model.fit()
print(fitted.summary())

# Forecast
fc = fitted.forecast(119, alpha=0.05)
fc_series = pd.Series(fc, index=test.index)
se = fitted.forecast(119, alpha=0.05)[1]
lower_series = pd.Series(fc - 1.96*se, index=test.index)
upper_series = pd.Series(fc + 1.96*se, 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()
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                  value   No. Observations:                   85
Model:                 ARIMA(3, 2, 1)   Log Likelihood                -116.886
Date:                Wed, 15 Mar 2023   AIC                            243.771
Time:                        09:37:25   BIC                            255.865
Sample:                    07-01-1991   HQIC                           248.630
                         - 07-01-1998
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.2205      0.194     -1.139      0.255      -0.600       0.159
ar.L2         -0.3436      0.141     -2.444      0.015      -0.619      -0.068
ar.L3         -0.0937      0.252     -0.371      0.710      -0.588       0.401
ma.L1         -0.9998     14.653     -0.068      0.946     -29.720      27.720
sigma2         0.9141     13.333      0.069      0.945     -25.219      27.047
===================================================================================
Ljung-Box (L1) (Q):                   0.09   Jarque-Bera (JB):                24.64
Prob(Q):                              0.76   Prob(JB):                         0.00
Heteroskedasticity (H):               1.53   Skew:                            -0.81
Prob(H) (two-sided):                  0.27   Kurtosis:                         5.12
===================================================================================

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




```
# Accuracy metrics
def forecast_accuracy(forecast, actual):
    mape = np.mean(np.abs(forecast - actual)/np.abs(actual))  # MAPE
    me = np.mean(forecast - actual)             # ME
    mae = np.mean(np.abs(forecast - actual))    # MAE
    mpe = np.mean((forecast - actual)/actual)   # MPE
    rmse = np.mean((forecast - actual)**2)**.5  # RMSE
    corr = np.corrcoef(forecast, actual)[0,1]   # corr
    mins = np.amin(np.hstack([forecast[:,None],
                              actual[:,None]]), axis=1)
    maxs = np.amax(np.hstack([forecast[:,None],
                              actual[:,None]]), axis=1)
    minmax = 1 - np.mean(mins/maxs)             # minmax
    acf1 = acf(fc-test)[1]                      # ACF1
    return({'mape':mape, 'me':me, 'mae': mae,
            'mpe': mpe, 'rmse':rmse, 'acf1':acf1,
            'corr':corr, 'minmax':minmax})

forecast_accuracy(fc, test.values)
```

```
{'mape': 0.23266899442329508,
 'me': -3.880721387967784,
 'mae': 3.9817524734401397,
 'mpe': -0.2193087744854135,
 'rmse': 5.307216001549148,
 'acf1': nan,
 'corr': 0.8736884585142838,
 'minmax': 0.23187003957555852}
```