```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

import statsmodels.api as sm
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.graphics.tsaplots import plot_pacf
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.ar_model import AR
from statsmodels.tsa.arima_model import ARMA, ARIMA
from statsmodels.tsa.statespace.sarimax import SARIMAX

from math import sqrt


import seaborn as sns

from random import random

from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error, median_absolute_error
```

```
df = pd.read_excel('passenger.xlsx',header=None)
```

```
df.columns = ['year','passengers']
```

```
df.head()
```

```
df.describe()
```

```
print('Time period start: {}\nTime period end: {}'.format(df.year.min(),df.year.max()))
```

```
    Time period start: 1949-01
    Time period end: 1960-12
```

```
df.shape
```

```
    (144, 2)
```

```
df['year'] = pd.to_datetime(df['year'], format='%Y-%m')
```

```
y = df.set_index('year')
```

```
y.index
```

```
    DatetimeIndex(['1949-01-01', '1949-02-01', '1949-03-01', '1949-04-01',
                   '1949-05-01', '1949-06-01', '1949-07-01', '1949-08-01',
                   '1949-09-01', '1949-10-01',
                   ...
                   '1960-03-01', '1960-04-01', '1960-05-01', '1960-06-01',
                   '1960-07-01', '1960-08-01', '1960-09-01', '1960-10-01',
                   '1960-11-01', '1960-12-01'],
                  dtype='datetime64[ns]', name='year', length=144, freq=None)
```

```
y.isnull().sum()
```

```
    passengers    0
    dtype: int64
```

```
y.plot(figsize=(15, 6))
plt.show()
```

```
plt.plot(y)
```

```
from statsmodels.tsa.stattools import adfuller
#H0:IT IS NON STATIONARY
#H1:IT IS  STATIONARY

#Perform Dickey-Fuller test:
print ('Results of Dickey-Fuller Test:')
dftest = adfuller(y.passengers)
dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags Used','Number of Observat
for key,value in dftest[4].items():
    dfoutput['Critical Value (%s)'%key] = value
print (dfoutput)
```

```
Results of Dickey-Fuller Test:
Test Statistic                   0.815369
p-value                          0.991880
#Lags Used                      13.000000
Number of Observations Used    130.000000
Critical Value (1%)             -3.481682
Critical Value (5%)             -2.884042
Critical Value (10%)            -2.578770
dtype: float64
```

```
def test_stationarity(timeseries):

    #Determing rolling statistics
    rolmean = timeseries.rolling(12).mean()
    rolstd = timeseries.rolling(12).std()

    #Plot rolling statistics:
    orig = plt.plot(timeseries, color='blue',label='Original')
    mean = plt.plot(rolmean, color='red', label='Rolling Mean')
    std = plt.plot(rolstd, color='black', label = 'Rolling Std')
    plt.legend(loc='best')
    plt.title('Rolling Mean & Standard Deviation')
    plt.show(block=False)

    #Perform Dickey-Fuller test:
    print ('Results of Dickey-Fuller Test:')
    dftest = adfuller(timeseries, autolag='AIC')
    dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags Used','Number of Obse
    for key,value in dftest[4].items():
```

```
        dfoutput['Critical Value (%s)'%key] = value
    print (dfoutput)

ts_log = np.log(y)  # USE LOG SCALE
plt.plot(ts_log)
```

```
ts_log_diff = ts_log.passengers - ts_log.passengers.shift(1)  # DIFFERENCING OREDER =1 ======ARIMA
plt.plot(ts_log_diff)
ts_log_diff.dropna(inplace=True)
test_stationarity(ts_log_diff)                           #stationary  series
```

```
from statsmodels.tsa.seasonal import seasonal_decompose
result = seasonal_decompose(ts_log_diff, model='additive',extrapolate_trend='freq')
result.plot()
plt.show()
```

```
from statsmodels.tsa.stattools import acf, pacf

lag_acf=acf(ts_log_diff, nlags=20)
lag_pacf=pacf(ts_log_diff, nlags=20, method='ols')

plt.figure(figsize=(20,10))
plt.subplot(121)
plt.plot(lag_acf)
plt.axhline(y=0,linestyle='--',color='green')
plt.axhline(y=-1.96/np.sqrt(len(ts_log_diff)),linestyle='--',color='green')
plt.axhline(y=1.96/np.sqrt(len(ts_log_diff)),linestyle='--',color='green')
plt.title('Autocorrelation Function')

plt.subplot(122)
plt.plot(lag_pacf)
plt.axhline(y=0,linestyle='--',color='green')
plt.axhline(y=-1.96/np.sqrt(len(ts_log_diff)),linestyle='--',color='green')
plt.axhline(y=1.96/np.sqrt(len(ts_log_diff)),linestyle='--',color='green')
plt.title('Autocorrelation Function')
```

```
actuals = ts_log[130:-1]
actuals
```

```
ts_log.shape
```

```
(144, 1)
```

```
import itertools
p= d = q = range(0, 4)
pdq = itertools.product(p,d,q)
for parameters in pdq:

  try:
    model=ARIMA(ts_log_diff , order=parameters)  #log transformation
    results=model.fit(disp=1)
    ypredicted = results.predict(130,142) # end point included
    mae = mean_absolute_error(actuals, ypredicted)
    print('ARIMA{} - MAE:{}'.format(parameters, mae))
    print('ARMA{} - AIC:{}'.format(parameters, results.aic))
  except:
    continue
```

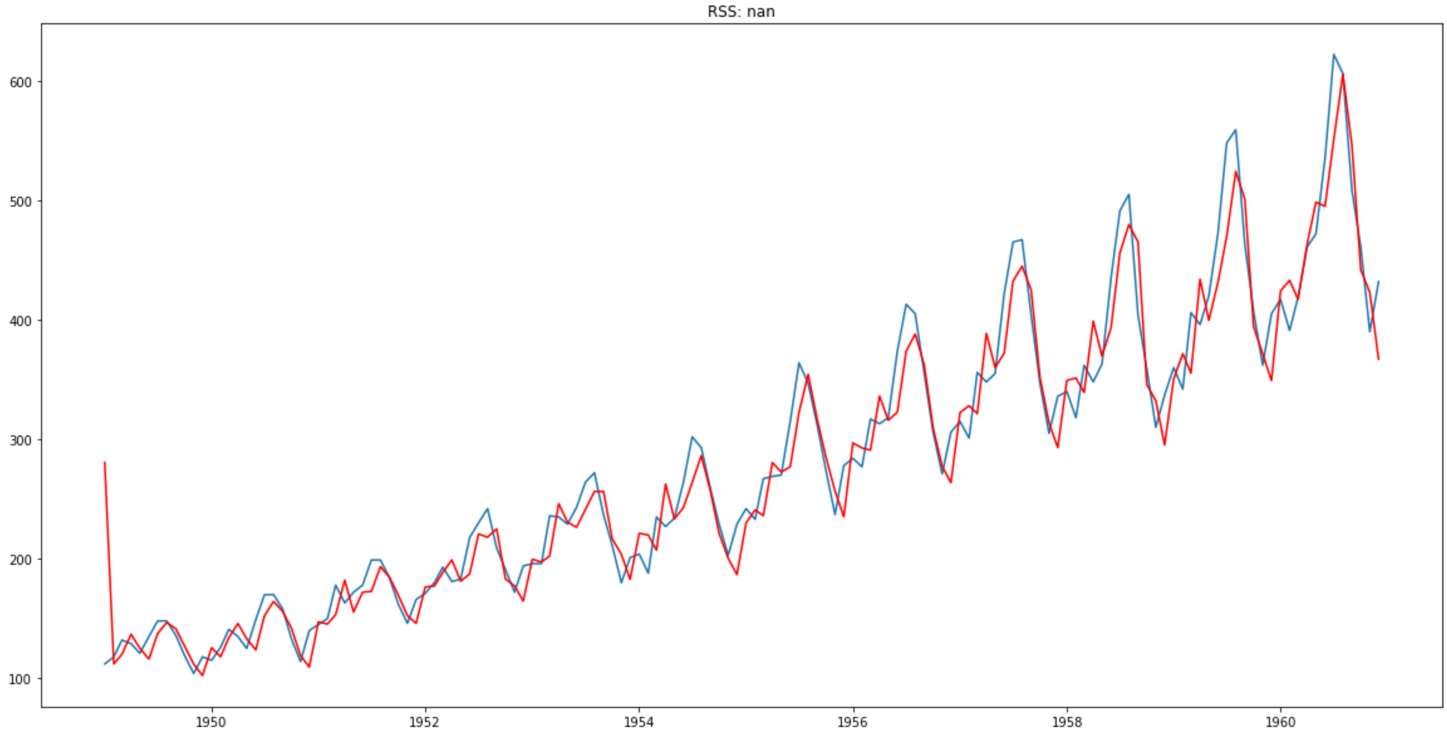
```
# finally:
#   print('ARIMA{} - MAE:{}'.format(parameters, mae))
```

```
ARIMA(3, 1, 1) - MAE:6.12010400547912
ARMA(3, 1, 1) - AIC:-227.33992067059341
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:165: ValueWarning:
  % freq, ValueWarning)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:165: ValueWarning:
  % freq, ValueWarning)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:668: RuntimeWarning: over
  newparams = ((1-np.exp(-params))/(1+np.exp(-params))).copy()
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:668: RuntimeWarning: inva
  newparams = ((1-np.exp(-params))/(1+np.exp(-params))).copy()
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:669: RuntimeWarning: over
  tmp = ((1-np.exp(-params))/(1+np.exp(-params))).copy()
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:669: RuntimeWarning: inva
  tmp = ((1-np.exp(-params))/(1+np.exp(-params))).copy()
ARIMA(3, 1, 2) - MAE:6.122809628798084
ARMA(3, 1, 2) - AIC:-228.75694269093208
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:165: ValueWarning:
  % freq, ValueWarning)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:165: ValueWarning:
  % freq, ValueWarning)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:165: ValueWarning:
  % freq, ValueWarning)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:165: ValueWarning:
  % freq, ValueWarning)
ARIMA(3, 2, 0) - MAE:6.131731487555997
ARMA(3, 2, 0) - AIC:-115.28613554782129
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:165: ValueWarning:
  % freq, ValueWarning)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:165: ValueWarning:
  % freq, ValueWarning)
ARIMA(3, 2, 1) - MAE:6.125843706180343
ARMA(3, 2, 1) - AIC:-173.220630090444
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:165: ValueWarning:
  % freq, ValueWarning)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:165: ValueWarning:
  % freq, ValueWarning)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:668: RuntimeWarning: over
  newparams = ((1-np.exp(-params))/(1+np.exp(-params))).copy()
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:668: RuntimeWarning: inva
  newparams = ((1-np.exp(-params))/(1+np.exp(-params))).copy()
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:669: RuntimeWarning: over
  tmp = ((1-np.exp(-params))/(1+np.exp(-params))).copy()
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:669: RuntimeWarning: inva
  tmp = ((1-np.exp(-params))/(1+np.exp(-params))).copy()
ARIMA(3, 2, 2) - MAE:6.125955871586712
ARMA(3, 2, 2) - AIC:-174.15822475241595

/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:165: ValueWarning:
  % freq, ValueWarning)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:165: ValueWarning:
  % freq, ValueWarning)
ARIMA(3, 2, 3) - MAE:6.11766906923324
ARMA(3, 2, 3) - AIC:-192.1088390701529
/usr/local/lib/python3.7/dist-packages/statsmodels/base/model.py:492: HessianInversionWarnin
  'available', HessianInversionWarning)
/usr/local/lib/python3.7/dist-packages/statsmodels/base/model.py:512: ConvergenceWarning: Ma
  "Check mle_retvals", ConvergenceWarning)
```

We chose the mdoel which has less MAE and more AUC (3,0,3)

```
from statsmodels.tsa.arima_model import ARIMA

plt.figure(figsize=(20,10))
model=ARIMA(y.passengers , order=(3,0,3))  #log transformation
results=model.fit(disp=1)
plt.plot(y.passengers)
plt.plot(results.fittedvalues, color='red')
plt.title('RSS: %.4f'% sum((results.fittedvalues-ts_log_diff)**2))
print('plotting ARIMA model')
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:165: ValueWarning: No
  % freq, ValueWarning)
/usr/local/lib/python3.7/dist-packages/statsmodels/base/model.py:512: ConvergenceWarning: Maxi
  "Check mle_retvals", ConvergenceWarning)
plotting ARIMA model
```




```
# model1=ARIMA(ts_log_diff , order=(3,0,3))#AR MODEL
# results1=model1.fit(disp=1)
plt.plot(y.passengers)
plt.plot(results.fittedvalues, color='red')
plt.title('RSS: %.4f'% sum((results.fittedvalues-ts_log_diff)**2))
print('plotting AR model')
```

```
predictions=pd.Series(results.fittedvalues, copy=True)
print(predictions.head())
```

```
year
1949-01-01    280.639144
1949-02-01    112.000085
1949-03-01    120.185185
1949-04-01    136.931162
1949-05-01    125.204261
dtype: float64
```

```
predictions_cum_sum=predictions.cumsum()
print(predictions_cum_sum.head())
```

```
year
1949-01-01    280.639144
1949-02-01    392.639229
1949-03-01    512.824414
1949-04-01    649.755576
1949-05-01    774.959837
dtype: float64
```

```
predictions_log=pd.Series(ts_log.passengers.iloc[0], index=ts_log.index)
predictions_log=predictions_log.add(predictions_cum_sum,fill_value=0)
predictions_log.head()
```

```
year
1949-01-01    285.357643
1949-02-01    397.357728
1949-03-01    517.542913
1949-04-01    654.474074
1949-05-01    779.678336
dtype: float64
```

```
results.plot_predict(1,264)    #144 + 12*10         (12*12 )+(12*10)
```

```
# Sarima=sm.tsa.statespace.SARIMAX(df['passengers'],order=(1,·0,·3),seasonal_order=(1,0,2,8))
#·Sarima_fit·=·Sarima.fit()
#·ypredicted·=·Sarima_fit.predict(len(df),·len(df)+12)··#·end·point·included
#·mae·=·mean_absolute_error(actuals,·ypredicted)
#·print('MAE:·%f'·%·mae)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/statespace/sarimax.py:949: UserWarning:
  warn('Non-stationary starting autoregressive parameters'
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/statespace/sarimax.py:981: UserWarning:
  warn('Non-stationary starting seasonal autoregressive'
MAE: 459.437236
```

```
def evaluate_forecast(y,pred):
    results = pd.DataFrame({'r2_score':r2_score(y, pred)}, index=[0])

    results['mean_absolute_error'] = mean_absolute_error(y, pred)
    results['median_absolute_error'] = median_absolute_error(y, pred)
    results['mse'] = mean_squared_error(y, pred)
    results['msle'] = mean_squared_log_error(y, pred)
    results['rmse'] = np.sqrt(results['mse'])
    return results
```

```
evaluate_forecast(actuals, ypredicted)    #actual          (y)    vs prediction(test)
```

0s completed at 9:54 PM