Forecasting number of Air Passengers for next 2 years

ARIMA and Seasonal ARIMA

Autoregressive Integrated Moving Averages

The general process for ARIMA models is the following:

- Visualize the Time Series Data
- Make the time series data stationary
- Plot the Correlation and AutoCorrelation Charts
- Construct the ARIMA Model or Seasonal ARIMA based on the data
- Use the model to make predictions

Let's go through these steps!

- Importing all the required libraries for analysis

In [51]:
```
import pandas as pd
import numpy as np
import matplotlib.pylab as plt
%matplotlib inline
from matplotlib.pylab import rcParams
from datetime import datetime
import warnings
warnings.filterwarnings('ignore')
```

- Reading the dataset file and checking the size of file

In [52]:
```
data=pd.read_csv(r'AirPassengers.csv')
data.shape
```

Out[52]:
```
(144, 2)
```

In [53]:
```
data.head()
```

Out[53]:

	Month	Passengers
0	1949-01	112
1	1949-02	118
2	1949-03	132
3	1949-04	129
4	1949-05	121

- Creating the 'Date' as Index for data and viewing the dataset

In [54]:
```
data.dtypes
```

Out[54]:
```
Month         object
Passengers     int64
dtype: object
```

In [55]:
```
# data['Month'] = pd.to_datetime(data['Month'], infer_datetime_format=True)
data['Month'] = pd.to_datetime(data['Month'], format="%Y-%m")
```

In [56]:
```
data.dtypes
```

Out[56]:
```
Month         datetime64[ns]
Passengers             int64
dtype: object
```

In [57]:
```
data.set_index(['Month'], inplace=True)
```

In [58]:
```
data.head()
```

Out[58]:

	Passengers
Month	
1949-01-01	112
1949-02-01	118
1949-03-01	132
1949-04-01	129
1949-05-01	121

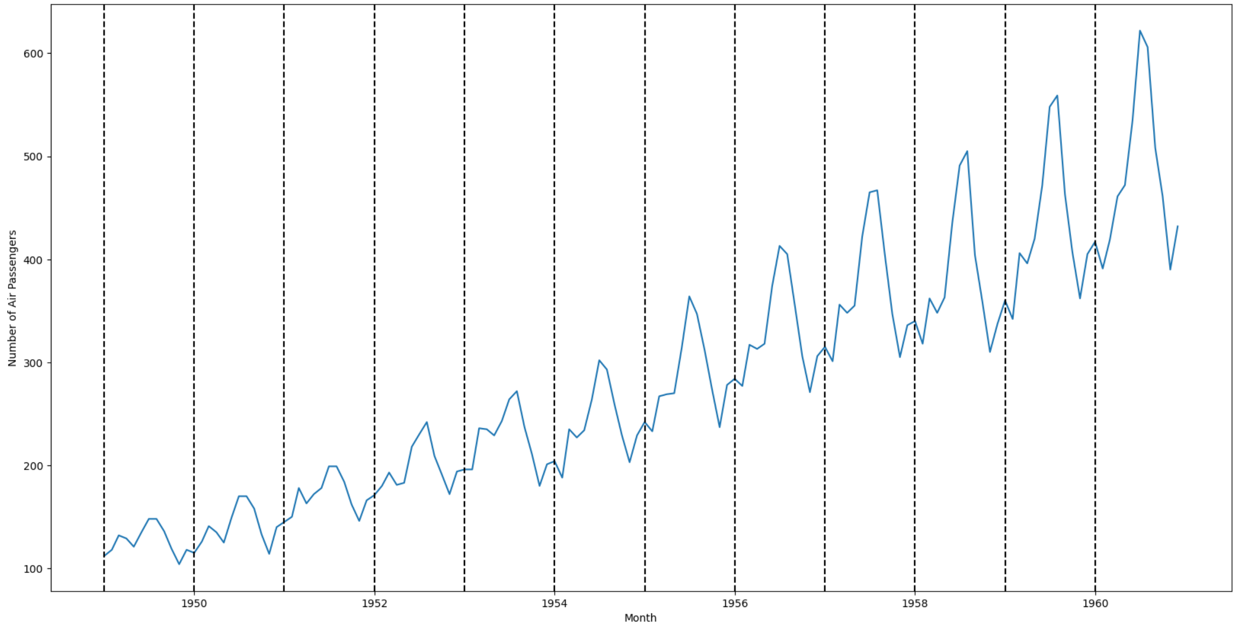
- Visualizing the Time Series plot for the number of Air Passengers

In [59]:
```
plt.figure(figsize=(20,10))
plt.xlabel("Month")
plt.ylabel("Number of Air Passengers")
plt.plot(data)

# dashed vertical line on start of each year
xcoords = ['1949-01-01', '1950-01-01', '1951-01-01', '1952-01-01', '1953-01-01',
           '1954-01-01', '1955-01-01', '1956-01-01', '1957-01-01', '1958-01-01',
```

```
          '1959-01-01', '1960-01-01']

for xc in xcoords:
    plt.axvline(x=pd.to_datetime(xc), color='black', linestyle='--')
```



Time Series Patterns

There are four types of time-series patterns:

- Trend: Long-term increase or decrease in the data. The trend can be any function, such as linear or exponential, and can change direction over time.
- Seasonality: Repeating cycle in the series with fixed frequencies (hour of the day, week, month, year, etc.). A seasonal pattern exists of a fixed known period.
- Cyclicity: Occurs when the data rise and fall, but without a fixed frequency and duration caused, for example, by economic conditions.
- Noise: The random variation in the series.

Stationarity

A stationary time series is one whose properties do not depend on the time at which the series is observed. Thus, time series with trends, or with seasonality, are not stationary — the trend and seasonality will affect the value of the time series at different times.

Mean, variance and other statistics of a stationary time series remains constant

Now we will check if our data is staionary, as ARIMA requires stationary data

Testing visually

In [60]:
```
rolmean = data.rolling(window=12).mean()
rolstd = data.rolling(window=12).std()
```

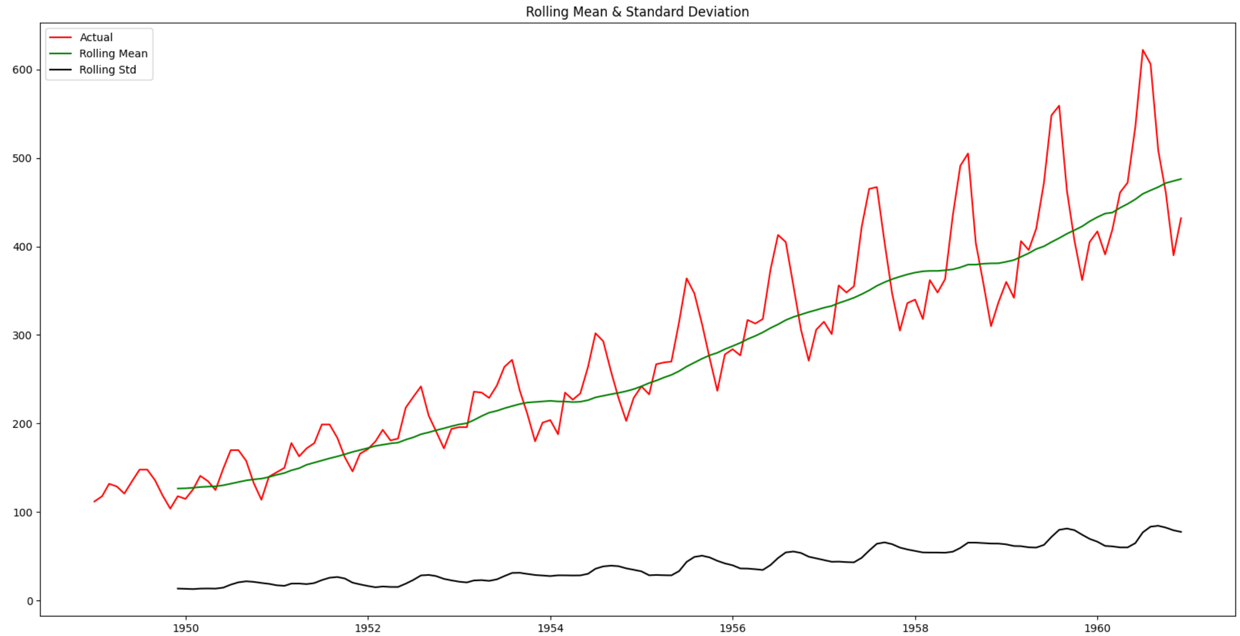
```
print(rolmean.head(15))
print(rolstd.head(15))
```

```
            Passengers
Month
1949-01-01         NaN
1949-02-01         NaN
1949-03-01         NaN
1949-04-01         NaN
1949-05-01         NaN
1949-06-01         NaN
1949-07-01         NaN
1949-08-01         NaN
1949-09-01         NaN
1949-10-01         NaN
1949-11-01         NaN
1949-12-01  126.666667
1950-01-01  126.916667
1950-02-01  127.583333
1950-03-01  128.333333
            Passengers
Month
1949-01-01         NaN
1949-02-01         NaN
1949-03-01         NaN
1949-04-01         NaN
1949-05-01         NaN
1949-06-01         NaN
1949-07-01         NaN
1949-08-01         NaN
1949-09-01         NaN
1949-10-01         NaN
1949-11-01         NaN
1949-12-01   13.720147
1950-01-01   13.453342
1950-02-01   13.166475
1950-03-01   13.686977
```

- Plotting the Rolling Mean and Standard Deviation, which has window of 12
- By looking below plot, we conclude that, it is non-stationary as mean and variance is not constant

In [61]:

```
plt.figure(figsize=(20,10))
actual = plt.plot(data, color='red', label='Actual')
mean_12 = plt.plot(rolmean, color='green', label='Rolling Mean')
std_12 = plt.plot(rolstd, color='black', label='Rolling Std')
plt.legend(loc='best')
plt.title('Rolling Mean & Standard Deviation')
plt.show(block=False)
```



Statistical testing

- ADF test
- KPSS test

ADF Testing

- Ho: It is non stationary
- H1: It is stationary

In [62]:
```
from statsmodels.tsa.stattools import adfuller
```

In [63]:
```
test_result = adfuller(data['Passengers'])
```

In [64]:
```
test_result
```

Out[64]:
```
(0.8153688792060447,
 0.9918802434376409,
 13,
 130,
 {'1%': -3.4816817173418295,
  '5%': -2.8840418343195267,
  '10%': -2.578770059171598},
 996.692930839019)
```

In [65]:
```
def adfuller_test(data):
    result = adfuller(data)
    labels = ['ADF Test Statistic', 'p-value', '#Lags Used', 'Number of Observations U
    for value, label in zip(result, labels):
        print(label+' : '+str(value) )
    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis(Ho), reject the null hypoth
    else:
        print("weak evidence against null hypothesis, time series has a unit root, ind
```

In [66]:
```
adfuller_test(data['Passengers'])
```

```
ADF Test Statistic : 0.8153688792060447
p-value : 0.9918802434376409
#Lags Used : 13
Number of Observations Used : 130
weak evidence against null hypothesis, time series has a unit root, indicating it is
non-stationary
```

- From above ADF test, we fail to reject the null hypothesis, since p-value is greater than 0.05

KPSS Testing

- Ho: It is stationary
- H1: It is non stationary

In [67]:
```
from statsmodels.tsa.stattools import kpss

def kpss_test(timeseries):
    print("Results of KPSS Test:")
    kpsstest = kpss(timeseries, regression="c")
    kpss_output = pd.Series(
        kpsstest[0:3], index=["Test Statistic", "p-value", "Lags Used"]
    )
    for key, value in kpsstest[3].items():
        kpss_output["Critical Value (%s)" % key] = value
    print(kpss_output)

    if (kpss_output['p-value'] < 0.05):
        print("The time series is not stationary")
    else:
        print("The time series is stationary")
```

In [68]:
```
kpss_test(data['Passengers'])
```

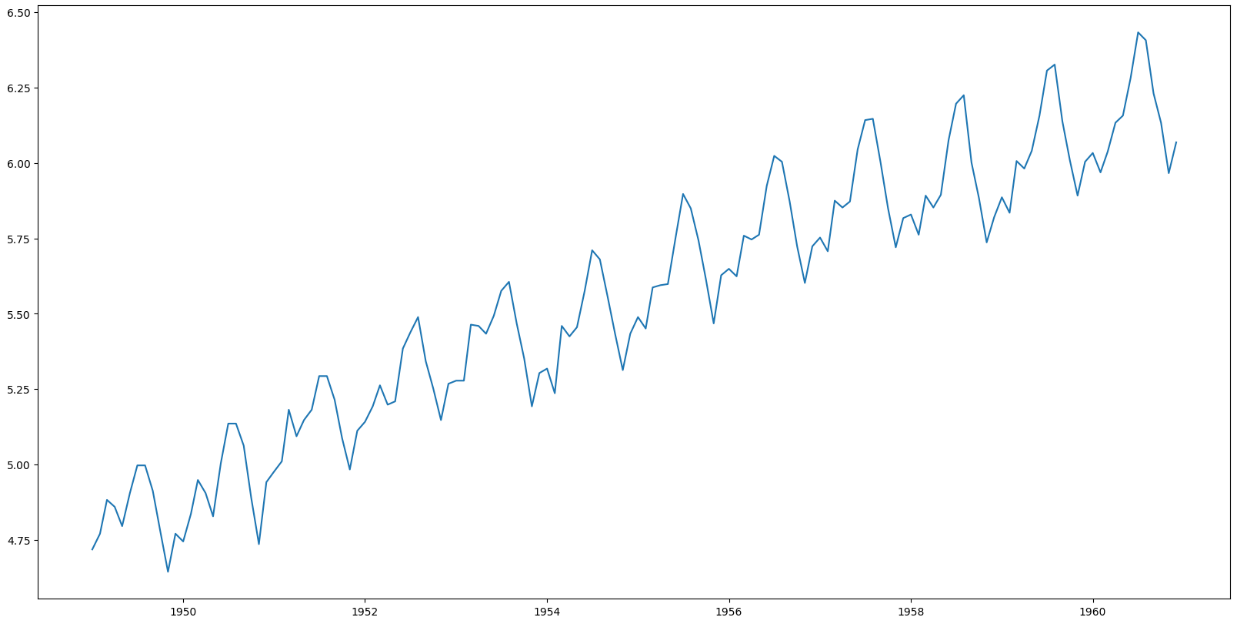
```
Results of KPSS Test:
Test Statistic           1.651312
p-value                  0.010000
Lags Used                8.000000
Critical Value (10%)     0.347000
Critical Value (5%)      0.463000
Critical Value (2.5%)    0.574000
Critical Value (1%)      0.739000
dtype: float64
The time series is not stationary
```

Log transformation

- Below we have taken log transformation to make variance stable and plotted visual for it
- We found graph upward trending over time with seasonality

In [69]:
```
plt.figure(figsize=(20,10))
data_log = np.log(data)
plt.plot(data_log)
```

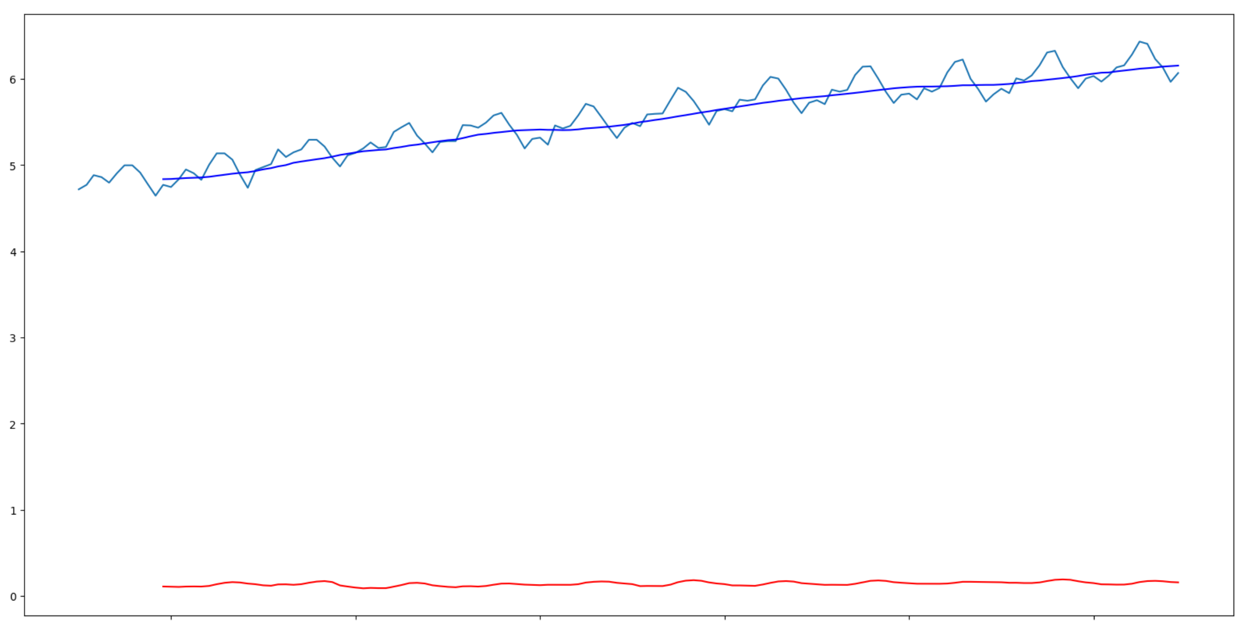
Out[69]: [<matplotlib.lines.Line2D at 0x14f31aa10>]



- Testing Rolling Mean with window 12 on above log transformation and concluded non-stationary, again

In [70]:
```
plt.figure(figsize=(20,10))
MAvg=data_log.rolling(window=12).mean()
MStd=data_log.rolling(window=12).std()
plt.plot(data_log)
plt.plot(MAvg, color='blue')
plt.plot(MStd, color='red')
```

Out[70]: [<matplotlib.lines.Line2D at 0x14f0ce350>]



The variance has stablized now. But the mean is still increasing.

In [71]:
```
adfuller_test(data_log['Passengers'])
```

```
ADF Test Statistic : -1.7170170891069627
p-value : 0.4223667747703902
#Lags Used : 13
Number of Observations Used : 130
weak evidence against null hypothesis, time series has a unit root, indicating it is
non-stationary
```

Not stationary yet.

In [72]:
```
kpss_test(data_log['Passengers'])
```

```
Results of KPSS Test:
Test Statistic            1.668651
p-value                   0.010000
Lags Used                 8.000000
Critical Value (10%)      0.347000
Critical Value (5%)       0.463000
Critical Value (2.5%)     0.574000
Critical Value (1%)       0.739000
dtype: float64
The time series is not stationary
```

Differencing to stabilize mean

In [73]:
```
data_log.head()
```

Out[73]:

Month	Passengers
1949-01-01	4.718499
1949-02-01	4.770685
1949-03-01	4.882802
1949-04-01	4.859812
1949-05-01	4.795791

In [74]:
```
data_log.shift().head()
```

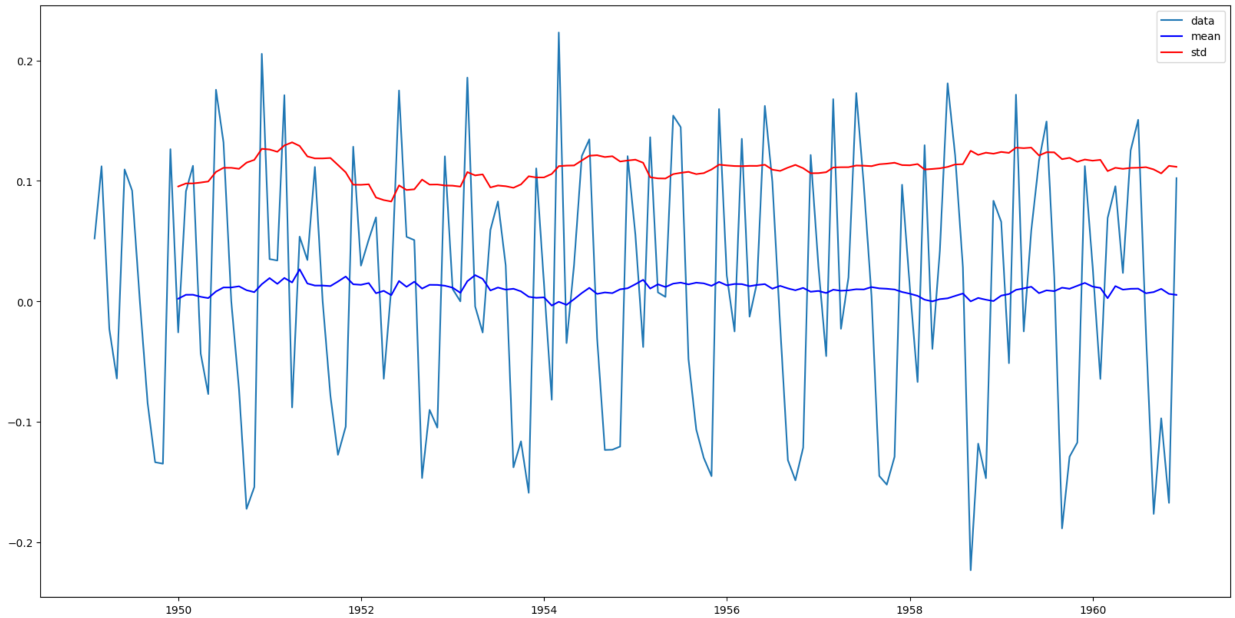
Out[74]:

Month	Passengers
1949-01-01	NaN
1949-02-01	4.718499
1949-03-01	4.770685
1949-04-01	4.882802
1949-05-01	4.859812

In [75]:
```
data_shift = data_log-data_log.shift()
```

In [76]:
```
plt.figure(figsize=(20,10))
MAvg=data_shift.rolling(window=12).mean()
MStd=data_shift.rolling(window=12).std()
plt.plot(data_shift, label='data')
plt.plot(MAvg, color='blue', label='mean')
plt.plot(MStd, color='red', label='std')
plt.legend()
```

Out[76]:
```
<matplotlib.legend.Legend at 0x14f1e4fd0>
```



In [77]:
```
adfuller_test(data_shift['Passengers'].dropna())
```

```
ADF Test Statistic : -2.7171305983881386
p-value : 0.07112054815085785
#Lags Used : 14
Number of Observations Used : 128
weak evidence against null hypothesis, time series has a unit root, indicating it is non-stationary
```

In [78]:
```
kpss_test(data_shift['Passengers'].dropna())
```

```
Results of KPSS Test:
Test Statistic            0.038304
p-value                   0.100000
Lags Used                 6.000000
Critical Value (10%)      0.347000
Critical Value (5%)       0.463000
Critical Value (2.5%)     0.574000
Critical Value (1%)       0.739000
dtype: float64
The time series is stationary
```

ARIMA Model
Auto Regressive Integrated Moving Average Model

p, d, q are used to characterize an ARIMA model

- p AR model lags

- d differencing

- q MA lags

Auto Regressive (AR) Model

AR.png

To find the order p of AR, we can use the pacf plot of the differenced data

Moving Average (MA) Model

MA.png

To find the order q of MA, we can use the acf plot of the differenced data

- Identification of an AR model is often best done with the PACF.
    - For an AR model, the theoretical PACF “shuts off” past the order of the model. The phrase “shuts off” means that in theory the partial autocorrelations are equal to 0 beyond that point. Put another way, the number of non-zero partial autocorrelations gives the order of the AR model. By the “order of the model” we mean the most extreme lag of x that is used as a predictor.

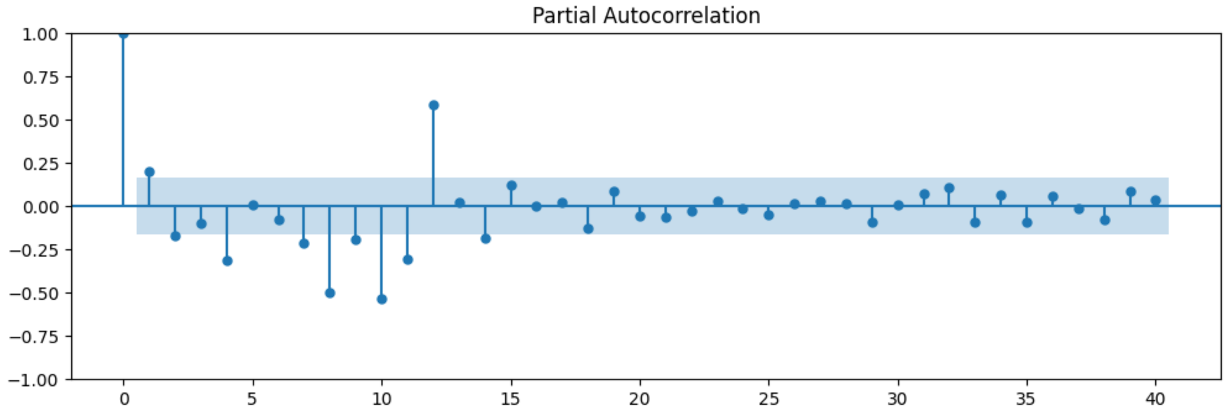
- Identification of an MA model is often best done with the ACF rather than the PACF.
    - For an MA model, the theoretical PACF does not shut off, but instead tapers toward 0 in some manner. A clearer pattern for an MA model is in the ACF. The ACF will have non-zero autocorrelations only at lags involved in the model.

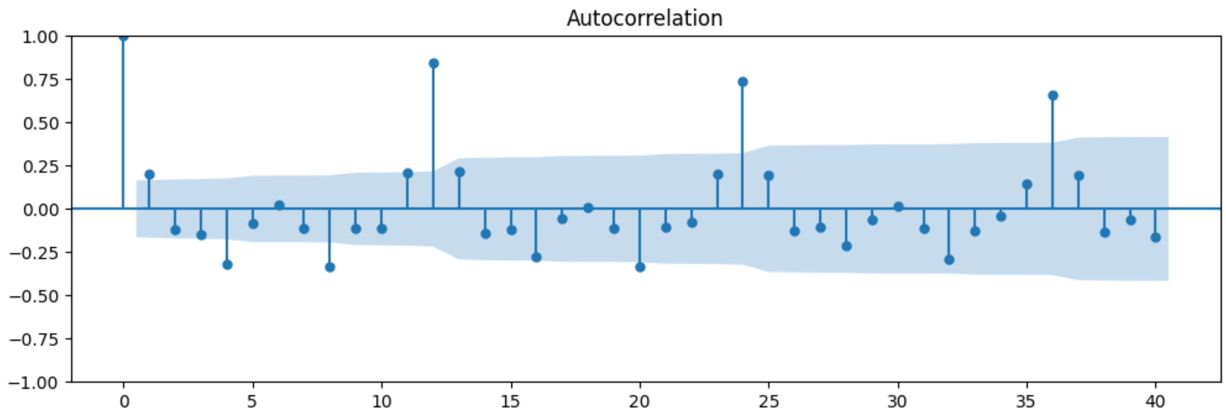
In [79]:

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

In [80]:

```
fig = plt.figure(figsize=(12,8))
ax1 = fig.add_subplot(211)
fig = plot_pacf(data_shift['Passengers'].dropna(), lags=40, ax=ax1)
ax2 = fig.add_subplot(212)
fig = plot_acf(data_shift['Passengers'].dropna(), lags=40, ax=ax2)
```





From the plots above, 1 seems like a good value for both p and q.

In [81]:
```
from statsmodels.tsa.arima.model import ARIMA
```

In [82]:
```
model = ARIMA(data_log['Passengers'], order=(1, 1, 1))
model_fit = model.fit()
```

Note that, we are using data_log as input instead of data_shift. It is because we are setting d=1 in ARIMA, therefore the ARIMA model will do the differencing itself.

In [83]:
```
model_fit.summary()
```

Out[83]:

SARIMAX Results

Dep. Variable:	Passengers	**No. Observations:**	144
Model:	ARIMA(1, 1, 1)	**Log Likelihood**	124.313
Date:	Thu, 30 Nov 2023	**AIC**	-242.626
Time:	16:48:18	**BIC**	-233.738
Sample:	01-01-1949	**HQIC**	-239.014
	- 12-01-1960		
Covariance Type:	opg		

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
ar.L1	-0.5773	0.164	-3.516	0.000	-0.899	-0.256
ma.L1	0.8478	0.098	8.685	0.000	0.656	1.039
sigma2	0.0103	0.002	5.992	0.000	0.007	0.014

Ljung-Box (L1) (Q):	0.02	**Jarque-Bera (JB):**	5.94
Prob(Q):	0.90	**Prob(JB):**	0.05
Heteroskedasticity (H):	1.07	**Skew:**	0.04
Prob(H) (two-sided):	0.82	**Kurtosis:**	2.00

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

In [84]:

```
data.shape
```

Out[84]:

```
(144, 1)
```

Prediction

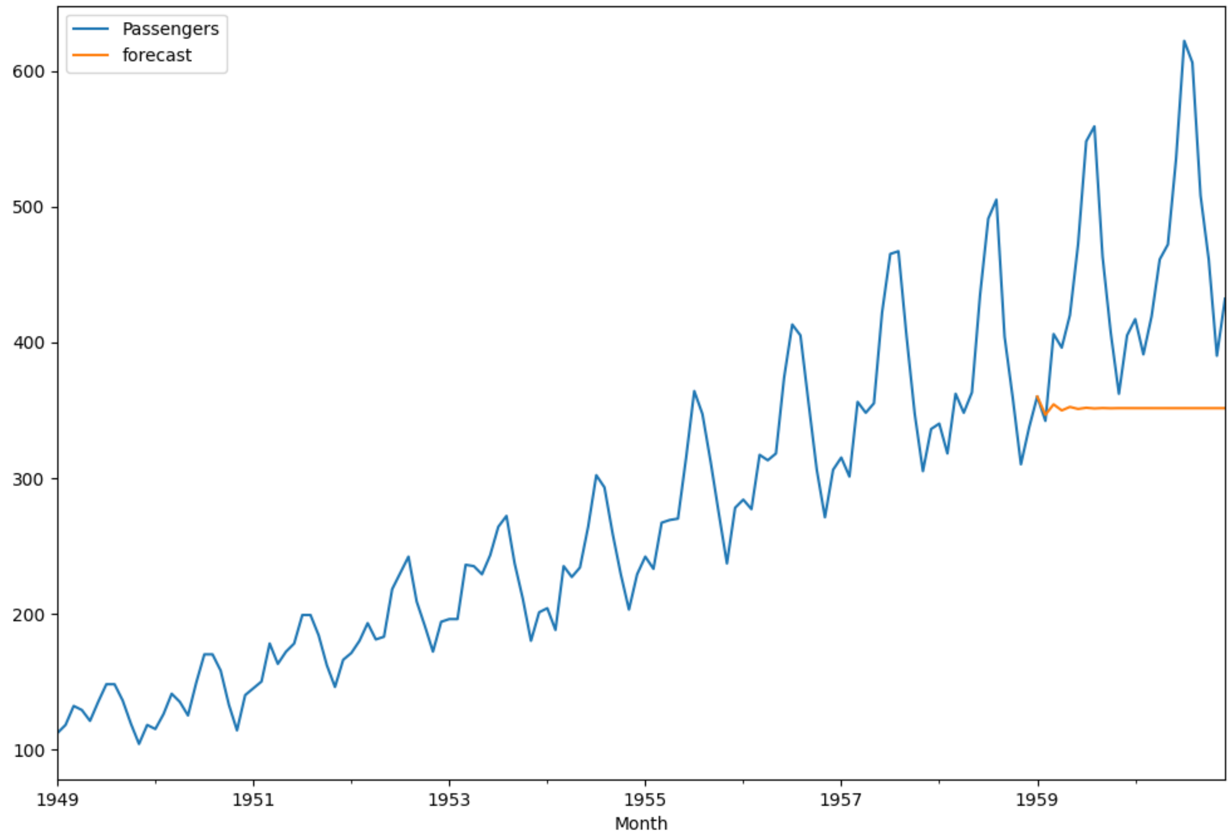
- In sample prediction
- Out of sample prediction

In [85]:

```
data['forecast'] = np.exp(model_fit.predict(start=120, end=144, dynamic=True))
data[['Passengers','forecast']].plot(figsize=(12,8))
```

Out[85]:

```
<Axes: xlabel='Month'>
```

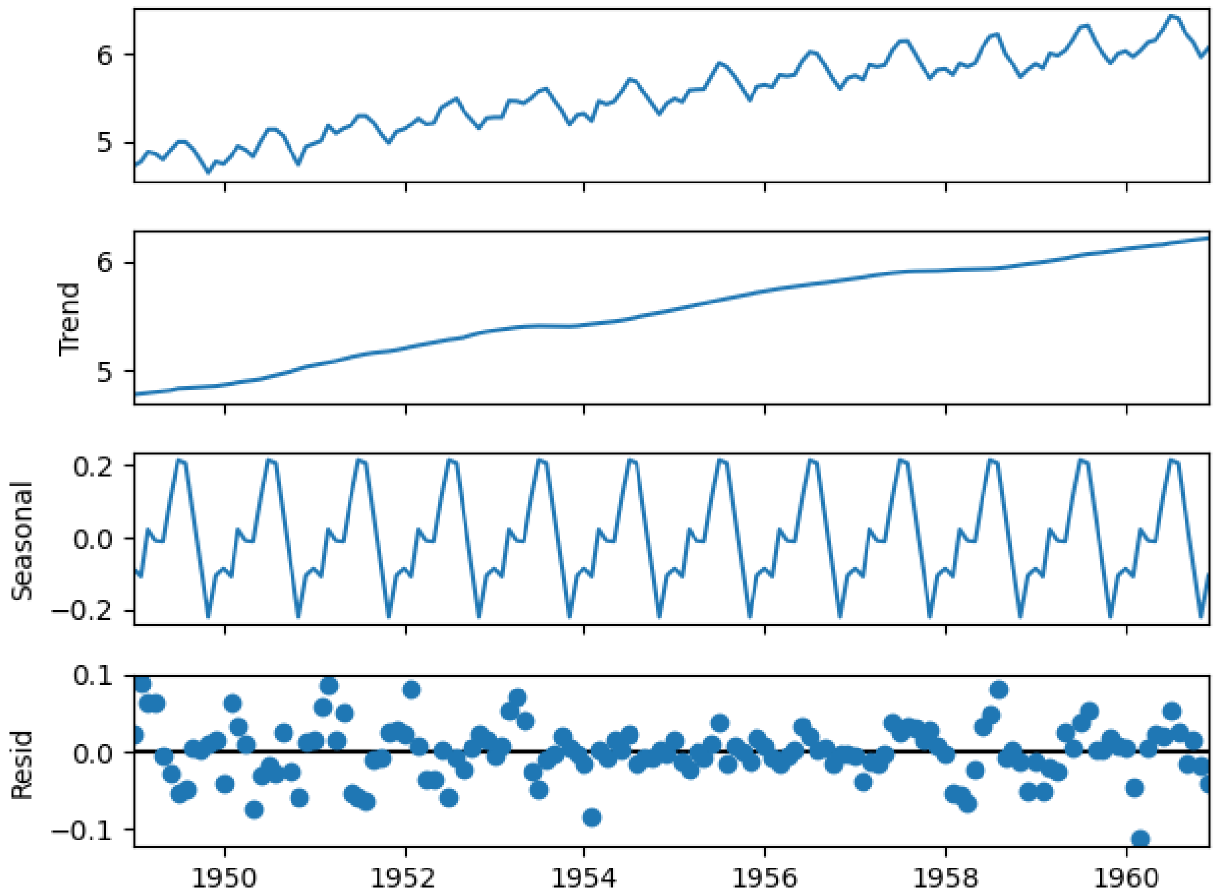


It is clear that the ARIMA model is not performing very well. We will now try the SARIMA model instead. It works well with seasonal data.

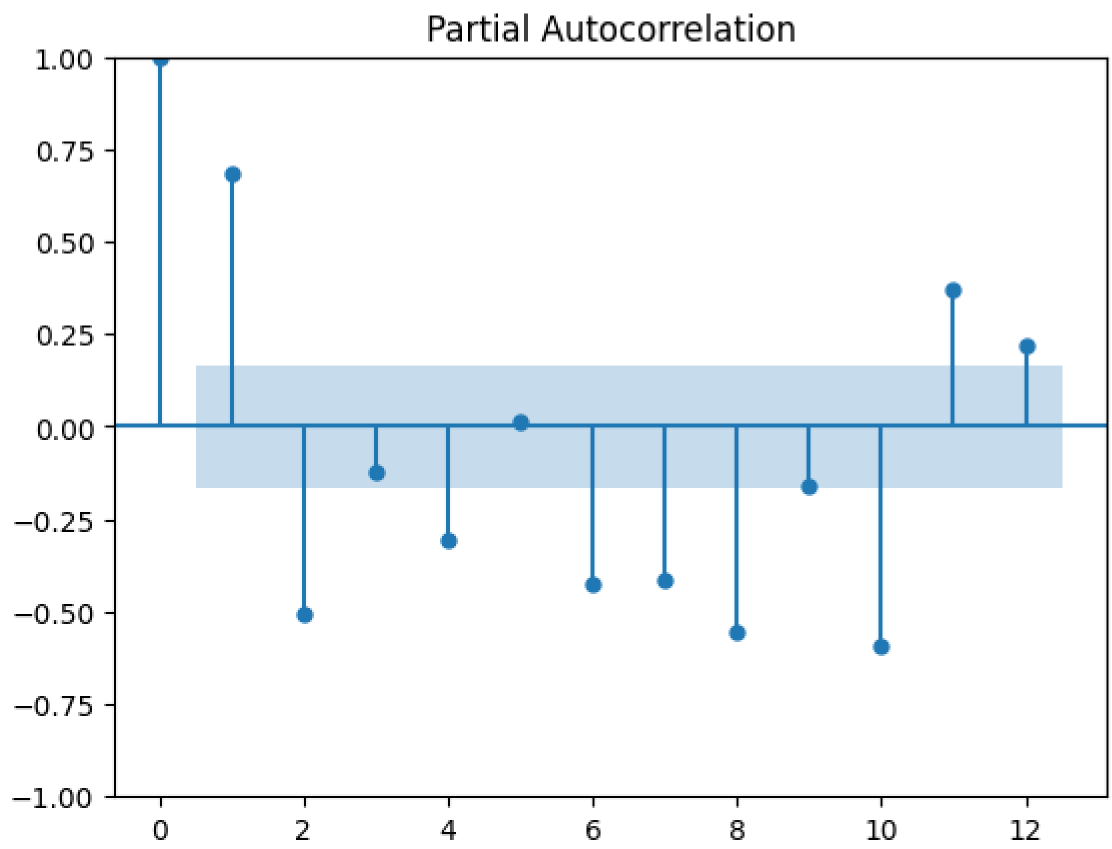
We need to define four more parameters: P, D, Q, and m.

- P indicates the Auto Regressive order for the seasonal component
- D indicates the integration order of the seasonal process (the number of transformation needed to make stationary the time series)
- Q indicated the Moving Average order for the seasonal component
- M indicates the periodicity, i.e. the number of periods in season, such as 12 for monthly data.

In [86]:
```
from statsmodels.tsa.seasonal import seasonal_decompose
result = seasonal_decompose(data_log, model='additive', extrapolate_trend='freq')
result.plot()
plt.show()
```

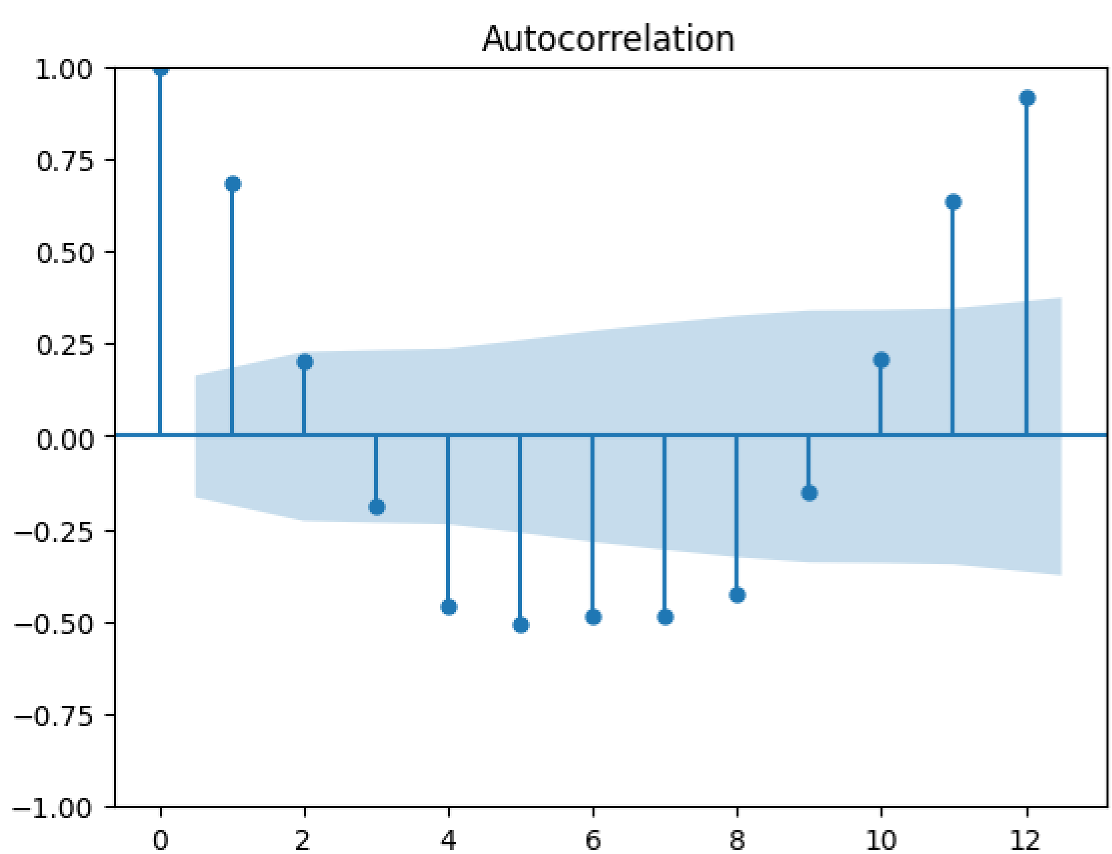


In [87]:
```
seasonal = result.seasonal
plot_pacf(seasonal, lags=12)
plt.show()
```



From the plot, 1 seems like a good value for P.

In [88]:
```
plot_acf(seasonal, lags=12)
plt.show()
```



From the plot, 1 seems like a good value for Q.

In [89]:
```
adfuller_test(seasonal.dropna())
```

```
ADF Test Statistic : -7238150499146665.0
p-value : 0.0
#Lags Used : 13
Number of Observations Used : 130
strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data has
no unit root and is stationary
```

As the seasonal data is stationary, D can be 0.

In [90]:
```
import statsmodels.api as sm
```

In [91]:
```
model2=sm.tsa.statespace.SARIMAX(data_log['Passengers'], order=(1, 1, 1), seasonal_orc
model_fit2=model2.fit()
```

```
RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =            5     M =           10

At X0         0 variables are exactly at the bounds

At iterate    0    f= -1.14164D+00    |proj g|=  7.85343D+00
 This problem is unconstrained.
```

```
At iterate    5    f= -1.75178D+00    |proj g|=  1.50730D+00

At iterate   10    f= -1.75311D+00    |proj g|=  6.12689D-02

At iterate   15    f= -1.75331D+00    |proj g|=  4.59452D-01

At iterate   20    f= -1.75345D+00    |proj g|=  3.33912D-02

At iterate   25    f= -1.75441D+00    |proj g|=  6.04289D-01

At iterate   30    f= -1.76443D+00    |proj g|=  1.10546D+00

At iterate   35    f= -1.76671D+00    |proj g|=  1.15980D-02

At iterate   40    f= -1.76671D+00    |proj g|=  7.91134D-03

At iterate   45    f= -1.76672D+00    |proj g|=  1.53358D-02

At iterate   50    f= -1.76677D+00    |proj g|=  8.01399D-02

           * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value

           * * *

   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
    5     50     69      1     0     0   8.014D-02  -1.767D+00
  F =  -1.7667746888209459

STOP: TOTAL NO. of ITERATIONS REACHED LIMIT
```

In [92]:

```
model_fit2.summary()
```

Out[92]:

SARIMAX Results

Dep. Variable:	Passengers	**No. Observations:**	144
Model:	SARIMAX(1, 1, 1)x(1, 0, 1, 12)	**Log Likelihood**	254.416
Date:	Thu, 30 Nov 2023	**AIC**	-498.831
Time:	16:48:20	**BIC**	-484.017
Sample:	01-01-1949	**HQIC**	-492.811
	- 12-01-1960		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.4120	0.166	2.477	0.013	0.086	0.738
ma.L1	-0.7272	0.134	-5.444	0.000	-0.989	-0.465
ar.S.L12	0.9869	0.010	99.800	0.000	0.968	1.006
ma.S.L12	-0.5546	0.111	-5.006	0.000	-0.772	-0.337
sigma2	0.0014	0.000	8.507	0.000	0.001	0.002

Ljung-Box (L1) (Q):	0.18	**Jarque-Bera (JB):**	3.91
Prob(Q):	0.67	**Prob(JB):**	0.14
Heteroskedasticity (H):	0.64	**Skew:**	0.03
Prob(H) (two-sided):	0.12	**Kurtosis:**	3.81

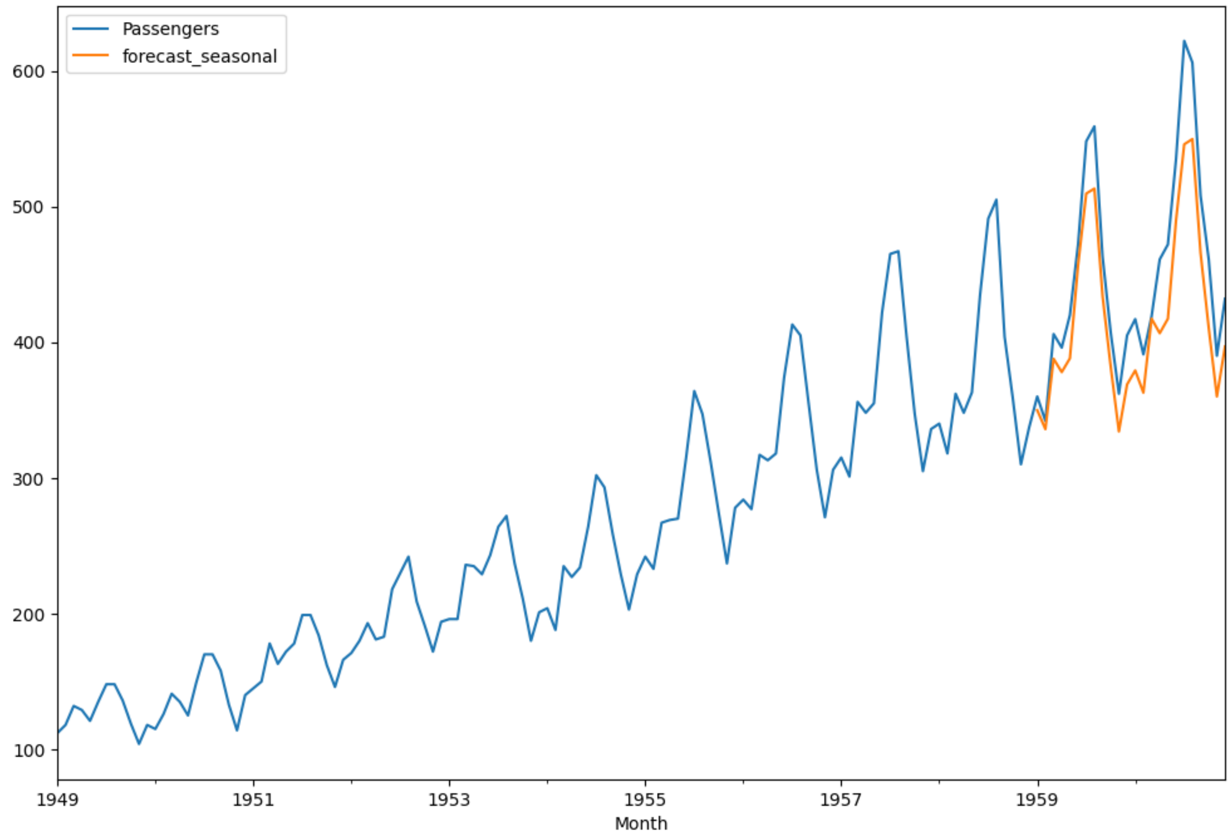
Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

In [93]:

```
data['forecast_seasonal'] = np.exp(model_fit2.predict(start=120, end=144, dynamic=True
data[['Passengers','forecast_seasonal']].plot(figsize=(12,8))
```

Out[93]:

```
<Axes: xlabel='Month'>
```



In [98]:
```
from statsmodels.tsa.stattools import acf
def forecast_accuracy(forecast, actual):
    forecast = np.array(forecast)
    actual = np.array(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(forecast-actual)[1]                      # ACF1
    return({'mape':mape, 'me':me, 'mae': mae,
            'mpe': mpe, 'rmse':rmse, 'acf1':acf1,
            'corr':corr, 'minmax':minmax})
```

In [95]:
```
forecast_accuracy(data['forecast_seasonal'][120:], data['Passengers'][120:])
```

Out[95]:
```
{'mape': 0.0724706827751485,
 'me': -33.94230725694194,
 'mae': 33.94230725694194,
 'mpe': -0.0724706827751485,
 'rmse': 38.114020419357225,
 'acf1': 0.4511701382592927,
 'corr': 0.9832032717146648,
 'minmax': 0.07247068277514845}
```

In [96]:
```
!pip install pmdarima
```

```
Requirement already satisfied: pmdarima in /Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages (2.0.4)
Requirement already satisfied: joblib>=0.11 in /Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages (from pmdarima) (1.2.0)
Requirement already satisfied: Cython!=0.29.18,!=0.29.31,>=0.29 in /Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages (from pmdarima) (3.0.6)
Requirement already satisfied: numpy>=1.21.2 in /Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages (from pmdarima) (1.21.6)
Requirement already satisfied: pandas>=0.19 in /Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages (from pmdarima) (1.3.5)
Requirement already satisfied: scikit-learn>=0.22 in /Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages (from pmdarima) (1.0.2)
Requirement already satisfied: scipy>=1.3.2 in /Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages (from pmdarima) (1.7.3)
Requirement already satisfied: statsmodels>=0.13.2 in /Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages (from pmdarima) (0.13.5)
Requirement already satisfied: urllib3 in /Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages (from pmdarima) (1.26.13)
Requirement already satisfied: setuptools!=50.0.0,>=38.6.0 in /Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages (from pmdarima) (65.6.3)
Requirement already satisfied: packaging>=17.1 in /Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages (from pmdarima) (21.3)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages (from packaging>=17.1->pmdarima) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7.3 in /Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages (from pandas>=0.19->pmdarima) (2.8.2)
Requirement already satisfied: pytz>=2017.3 in /Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages (from pandas>=0.19->pmdarima) (2022.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in /Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages (from scikit-learn>=0.22->pmdarima) (3.1.0)
Requirement already satisfied: patsy>=0.5.2 in /Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages (from statsmodels>=0.13.2->pmdarima) (0.5.3)
Requirement already satisfied: six in /Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages (from patsy>=0.5.2->statsmodels>=0.13.2->pmdarima) (1.16.0)

[notice] A new release of pip is available: 23.2 -> 23.3.1
[notice] To update, run: python -m pip install --upgrade pip
```

In [97]:

```python
import pmdarima as pm
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
/Users/sayemhasan/Downloads/Teaching/IAC/BA/Lec 13 Time Series Solution/Air_Passenger_Updated.ipynb Cell 80 line 1
----> <a href='vscode-notebook-cell:/Users/sayemhasan/Downloads/Teaching/IAC/BA/Lec%2013%20Time%20Series%20Solution/Air_Passenger_Updated.ipynb#Y142sZmlsZQ%3D%3D?line=0'>1</a> import pmdarima as pm

ModuleNotFoundError: No module named 'pmdarima'
```

In [ ]:

```python
smodel = pm.auto_arima(data_log['Passengers'], start_p=0, start_q=0,
                        test='adf',
                        max_p=3, max_q=3, m=12,
                        start_P=0, seasonal=True,
                        d=None, D=None, trace=True,
                        error_action='ignore',
```

```
                         suppress_warnings=True,
                         stepwise=True)
```

```
Performing stepwise search to minimize aic
 ARIMA(0,1,0)(0,1,1)[12]             : AIC=-467.553, Time=0.21 sec
 ARIMA(0,1,0)(0,1,0)[12]             : AIC=-434.830, Time=0.06 sec
 ARIMA(1,1,0)(1,1,0)[12]             : AIC=-474.808, Time=0.14 sec
 ARIMA(0,1,1)(0,1,1)[12]             : AIC=-483.393, Time=0.39 sec
 ARIMA(0,1,1)(0,1,0)[12]             : AIC=-449.978, Time=0.07 sec
 ARIMA(0,1,1)(1,1,1)[12]             : AIC=-481.906, Time=0.64 sec
 ARIMA(0,1,1)(0,1,2)[12]             : AIC=-481.956, Time=0.58 sec
 ARIMA(0,1,1)(1,1,0)[12]             : AIC=-477.399, Time=0.26 sec
 ARIMA(0,1,1)(1,1,2)[12]             : AIC=-479.902, Time=0.62 sec
 ARIMA(1,1,1)(0,1,1)[12]             : AIC=-481.893, Time=0.60 sec
 ARIMA(0,1,2)(0,1,1)[12]             : AIC=-481.610, Time=0.42 sec
 ARIMA(1,1,0)(0,1,1)[12]             : AIC=-481.484, Time=0.40 sec
 ARIMA(1,1,2)(0,1,1)[12]             : AIC=-479.399, Time=0.64 sec
 ARIMA(0,1,1)(0,1,1)[12] intercept   : AIC=-481.421, Time=0.47 sec

Best model:  ARIMA(0,1,1)(0,1,1)[12]
Total fit time: 5.543 seconds
```

In [ ]:

```
smodel = pm.auto_arima(data_log['Passengers'], start_p=0, start_q=0,
                         test='adf',
                         max_p=3, max_q=3, m=12,
                         start_P=0, seasonal=True,
                         d=None, D=None, trace=True,
                         error_action='ignore',
                         suppress_warnings=True,
                         stepwise=True)
```

```
Performing stepwise search to minimize aic
 ARIMA(0,1,0)(0,1,1)[12]             : AIC=-467.553, Time=0.20 sec
 ARIMA(0,1,0)(0,1,0)[12]             : AIC=-434.830, Time=0.07 sec
 ARIMA(1,1,0)(1,1,0)[12]             : AIC=-474.808, Time=0.13 sec
 ARIMA(0,1,1)(0,1,1)[12]             : AIC=-483.393, Time=0.39 sec
 ARIMA(0,1,1)(0,1,0)[12]             : AIC=-449.978, Time=0.07 sec
 ARIMA(0,1,1)(1,1,1)[12]             : AIC=-481.906, Time=0.60 sec
 ARIMA(0,1,1)(0,1,2)[12]             : AIC=-481.956, Time=0.58 sec
 ARIMA(0,1,1)(1,1,0)[12]             : AIC=-477.399, Time=0.23 sec
 ARIMA(0,1,1)(1,1,2)[12]             : AIC=-479.902, Time=0.65 sec
 ARIMA(1,1,1)(0,1,1)[12]             : AIC=-481.893, Time=0.65 sec
 ARIMA(0,1,2)(0,1,1)[12]             : AIC=-481.610, Time=0.46 sec
 ARIMA(1,1,0)(0,1,1)[12]             : AIC=-481.484, Time=0.42 sec
 ARIMA(1,1,2)(0,1,1)[12]             : AIC=-479.399, Time=0.66 sec
 ARIMA(0,1,1)(0,1,1)[12] intercept   : AIC=-481.421, Time=0.52 sec

Best model:  ARIMA(0,1,1)(0,1,1)[12]
Total fit time: 5.654 seconds
```

In [ ]:

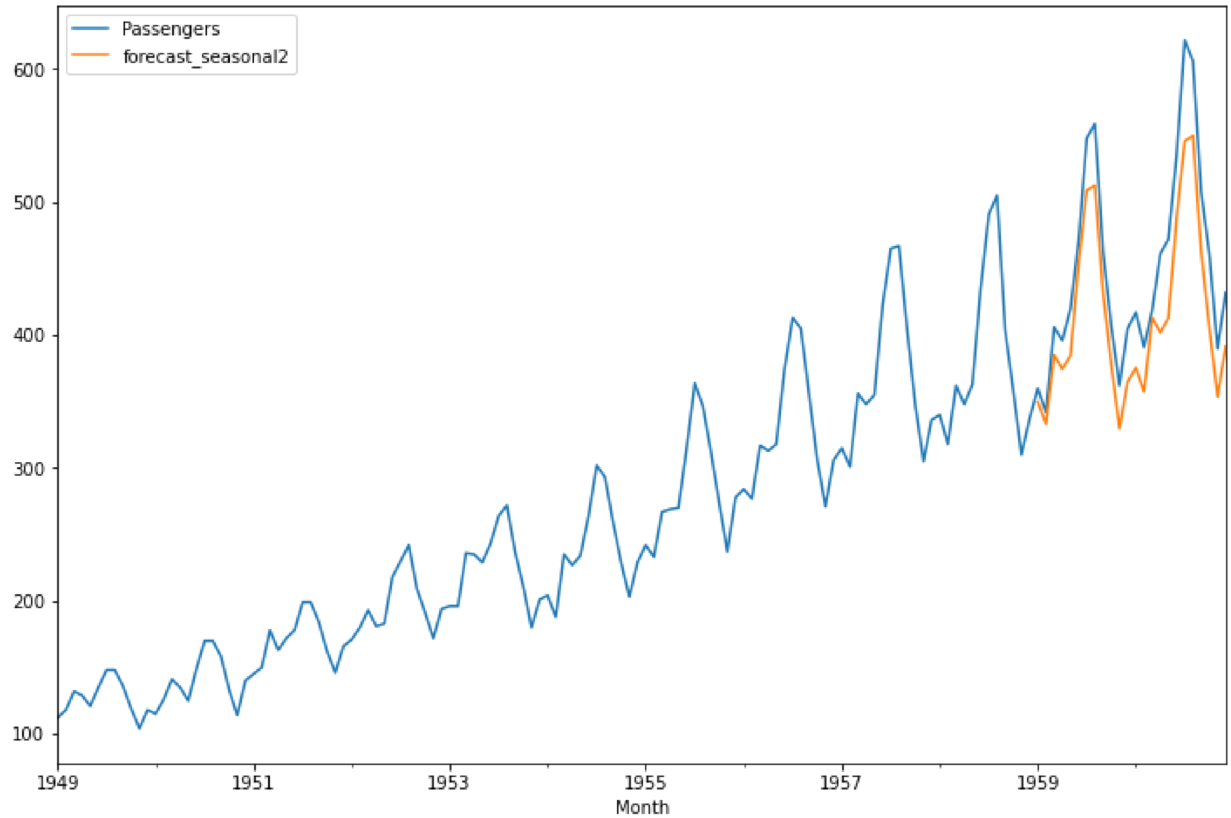
```
model2=sm.tsa.statespace.SARIMAX(data_log['Passengers'], order=(0, 1, 1), seasonal_orc
model_fit2=model2.fit()
```

In [ ]:

```
data['forecast_seasonal2'] = np.exp(model_fit2.predict(start=120, end=144, dynamic=Tru
data[['Passengers','forecast_seasonal2']].plot(figsize=(12,8))
```

Out[ ]:

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



In [ ]:
```
forecast_accuracy(data['forecast_seasonal'][120:], data['Passengers'][120:])
```

Out[ ]:
```
{'mape': 0.06729599033801562,
 'me': -31.455834595250696,
 'mae': 31.506020380116638,
 'mpe': -0.06717621519513747,
 'rmse': 35.57755931126885,
 'acf1': 0.4082402170799098,
 'corr': 0.9835898434270398,
 'minmax': 0.06729590438504685}
```

Forecasting future data

In [ ]:
```
data.head()
```

Out[ ]:

Month	Passengers	forecast	forecast_seasonal	forecast_seasonal2
1949-01-01	112	NaN	NaN	NaN
1949-02-01	118	NaN	NaN	NaN
1949-03-01	132	NaN	NaN	NaN
1949-04-01	129	NaN	NaN	NaN
1949-05-01	121	NaN	NaN	NaN

In [ ]:
```
from pandas.tseries.offsets import DateOffset
future_dates=[data.index[-1] + DateOffset(months=x) for x in range(0, 24)]
```

In [ ]:
```
data.index[-1]
```

Out[ ]:
```
Timestamp('1960-12-01 00:00:00')
```

In [ ]:
```
from pandas.tseries.offsets import DateOffset

future_dates = []
for i in range(24):
    future_dates.append(data.index[-1] + DateOffset(months=i+1))
```

In [ ]:
```
future_dates[:5]
```

Out[ ]:
```
[Timestamp('1961-01-01 00:00:00'),
 Timestamp('1961-02-01 00:00:00'),
 Timestamp('1961-03-01 00:00:00'),
 Timestamp('1961-04-01 00:00:00'),
 Timestamp('1961-05-01 00:00:00')]
```

In [ ]:
```
future_data_df = pd.DataFrame(index=future_dates, columns=data.columns)
```

In [ ]:
```
future_data_df.head()
```

Out[ ]:

	Passengers	forecast	forecast_seasonal	forecast_seasonal2
1961-01-01	NaN	NaN	NaN	NaN
1961-02-01	NaN	NaN	NaN	NaN
1961-03-01	NaN	NaN	NaN	NaN
1961-04-01	NaN	NaN	NaN	NaN
1961-05-01	NaN	NaN	NaN	NaN

In [ ]:
```
future_df = pd.concat([data, future_data_df])
```

In [ ]:
```
future_df.tail(30)
```

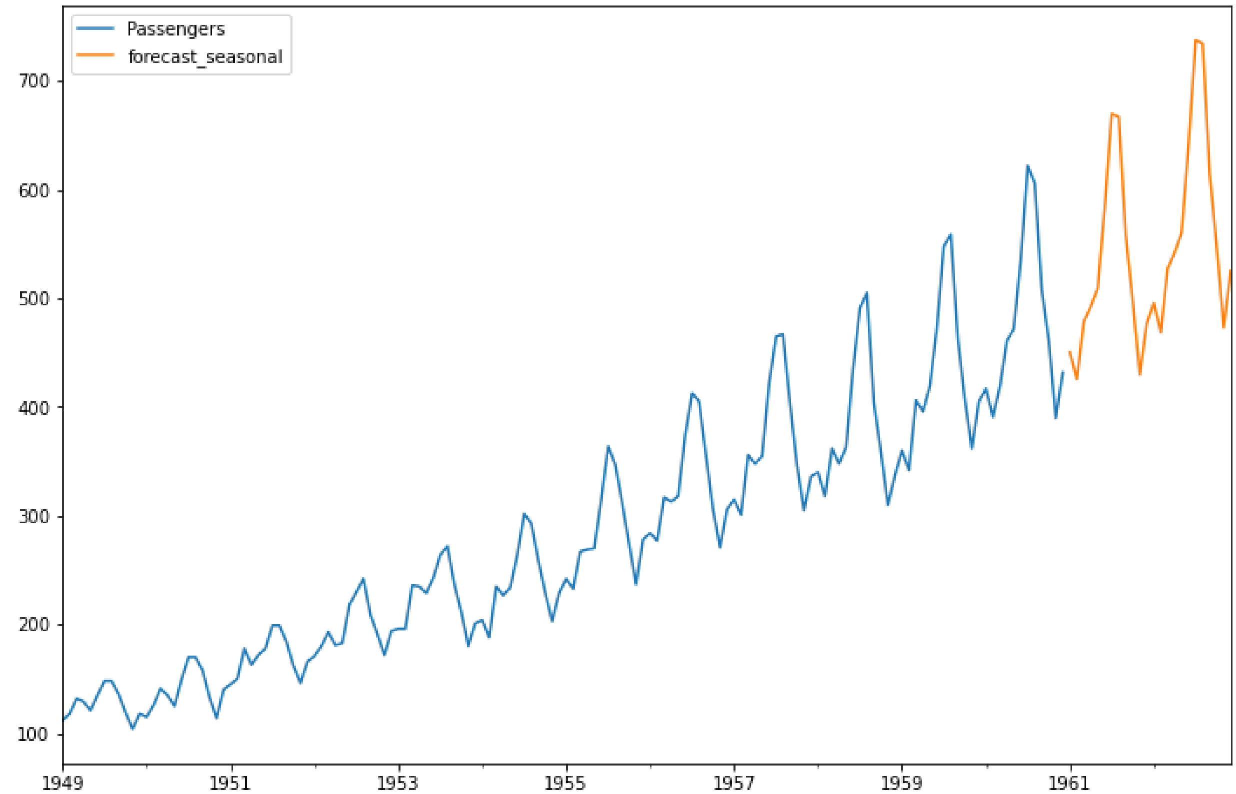
Out[ ]:

	Passengers	forecast	forecast_seasonal	forecast_seasonal2
1960-07-01	622	351.362295	551.032737	545.998373
1960-08-01	606	351.361618	555.139851	549.969332
1960-09-01	508	351.362009	469.156769	462.781958
1960-10-01	461	351.361783	412.801894	405.655523
1960-11-01	390	351.361913	361.546629	353.709954
1960-12-01	432	351.361838	398.479683	391.560918
1961-01-01	NaN	NaN	NaN	NaN
1961-02-01	NaN	NaN	NaN	NaN
1961-03-01	NaN	NaN	NaN	NaN
1961-04-01	NaN	NaN	NaN	NaN
1961-05-01	NaN	NaN	NaN	NaN
1961-06-01	NaN	NaN	NaN	NaN
1961-07-01	NaN	NaN	NaN	NaN
1961-08-01	NaN	NaN	NaN	NaN
1961-09-01	NaN	NaN	NaN	NaN
1961-10-01	NaN	NaN	NaN	NaN
1961-11-01	NaN	NaN	NaN	NaN
1961-12-01	NaN	NaN	NaN	NaN
1962-01-01	NaN	NaN	NaN	NaN
1962-02-01	NaN	NaN	NaN	NaN
1962-03-01	NaN	NaN	NaN	NaN
1962-04-01	NaN	NaN	NaN	NaN
1962-05-01	NaN	NaN	NaN	NaN
1962-06-01	NaN	NaN	NaN	NaN
1962-07-01	NaN	NaN	NaN	NaN
1962-08-01	NaN	NaN	NaN	NaN
1962-09-01	NaN	NaN	NaN	NaN
1962-10-01	NaN	NaN	NaN	NaN
1962-11-01	NaN	NaN	NaN	NaN
1962-12-01	NaN	NaN	NaN	NaN

In [ ]:

```
future_df['forecast_seasonal'] = np.exp(model_fit2.forecast(24))
future_df[['Passengers', 'forecast_seasonal']].plot(figsize=(12, 8))
```

Out[ ]: <AxesSubplot:>



Passengers
forecast_seasonal
700
600
500
400
300
200
100
1949
1951
1953
1955
1957
1959
1961

************************End***************************