Time series forecasting

Initial setup

Import required packages

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

import warnings
warnings.filterwarnings("ignore")
```

Import time series data: Airline passenger traffic

In [2]:
```
data = pd.read_csv('airline-passenger-traffic.csv', header = None)
data.columns = ['Month','Passengers']
data['Month'] = pd.to_datetime(data['Month'], format='%Y-%m')
data = data.set_index('Month')
data.head(12)
```

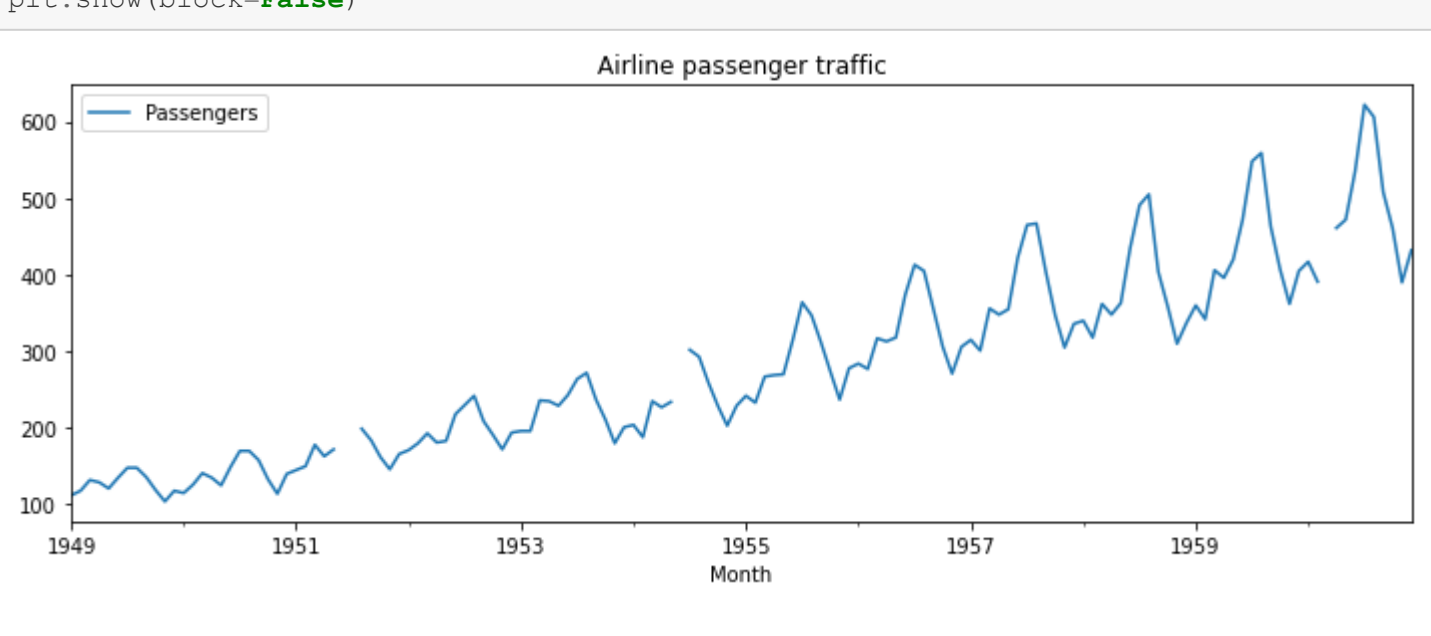
Out[2]:

Month	Passengers
1949-01-01	112.0
1949-02-01	118.0
1949-03-01	132.0
1949-04-01	129.0
1949-05-01	121.0
1949-06-01	135.0
1949-07-01	148.0
1949-08-01	148.0
1949-09-01	136.0
1949-10-01	119.0
1949-11-01	104.0
1949-12-01	118.0

Time series analysis

Plot time series data

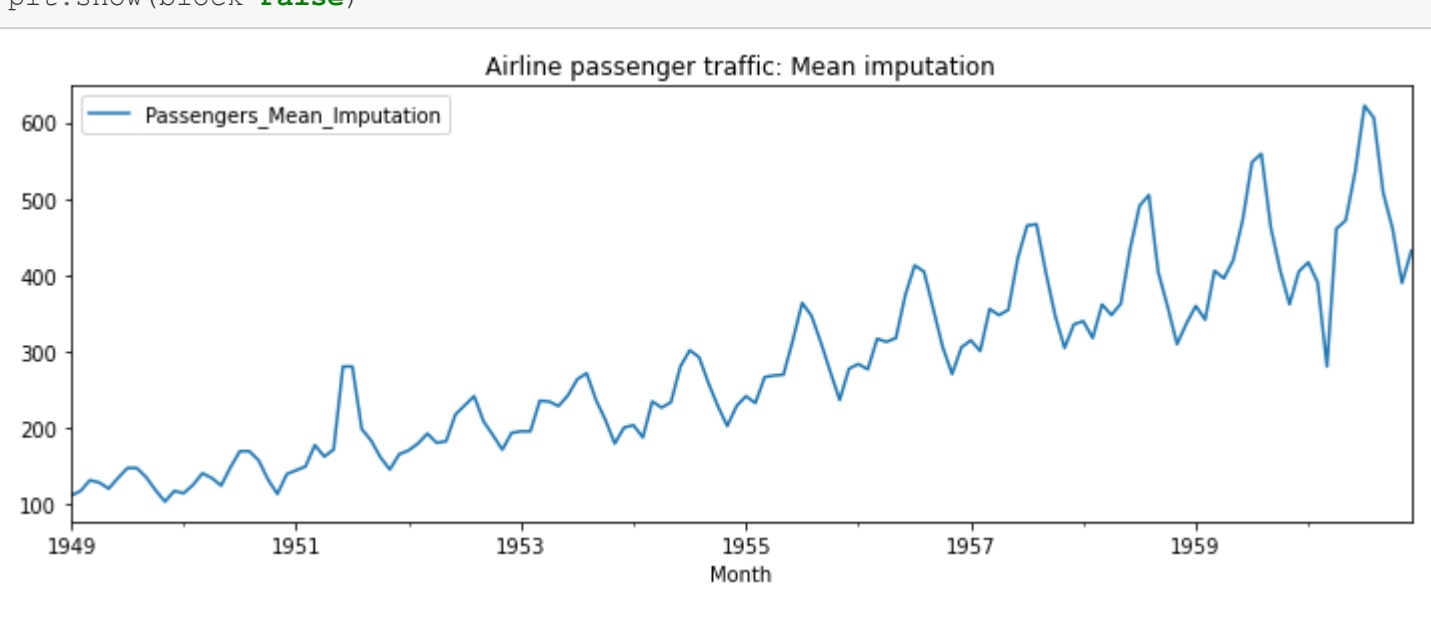
In [3]:
```
data.plot(figsize=(12, 4))
plt.legend(loc='best')
plt.title('Airline passenger traffic')
plt.show(block=False)
```



Missing value treatment

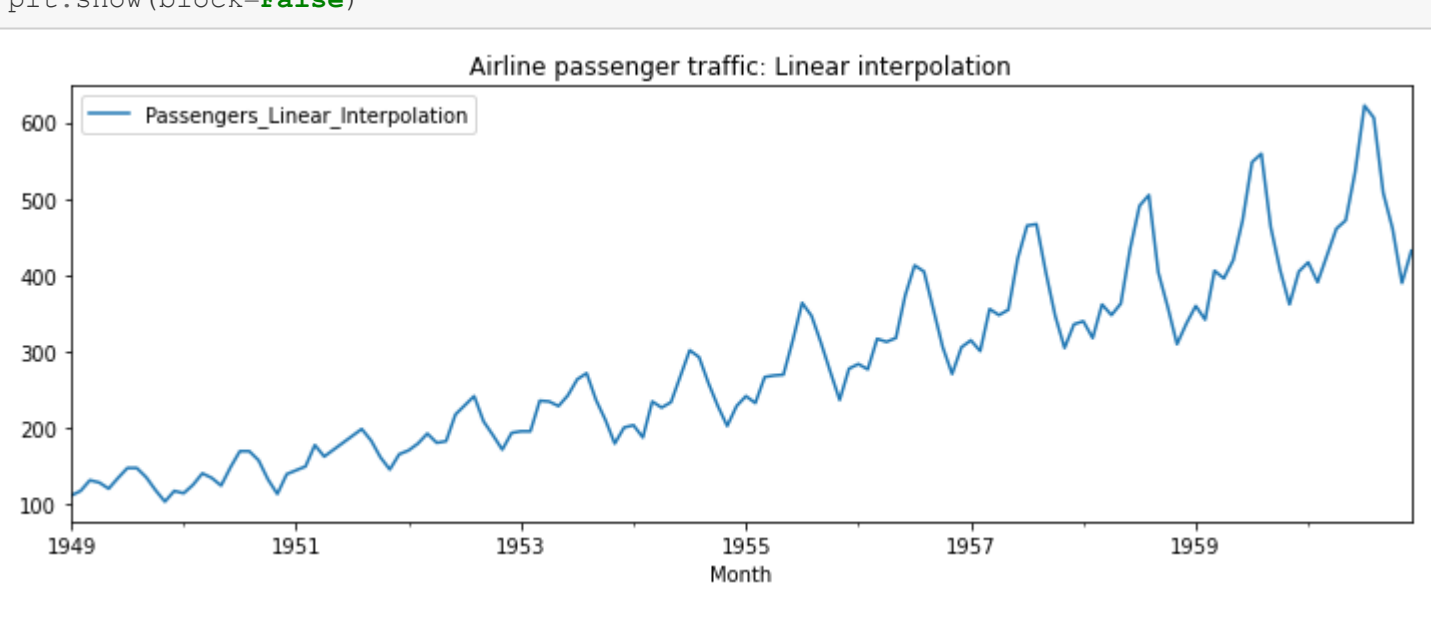
Mean imputation

In [4]:
```
data = data.assign(Passengers_Mean_Imputation=data.Passengers.fillna(data.Passengers.mean()))
data[['Passengers_Mean_Imputation']].plot(figsize=(12, 4))
plt.legend(loc='best')
plt.title('Airline passenger traffic: Mean imputation')
plt.show(block=False)
```



Linear interpolation

In [5]:
```
data = data.assign(Passengers_Linear_Interpolation=data.Passengers.interpolate(method='linear'))
data[['Passengers_Linear_Interpolation']].plot(figsize=(12, 4))
plt.legend(loc='best')
plt.title('Airline passenger traffic: Linear interpolation')
plt.show(block=False)
```



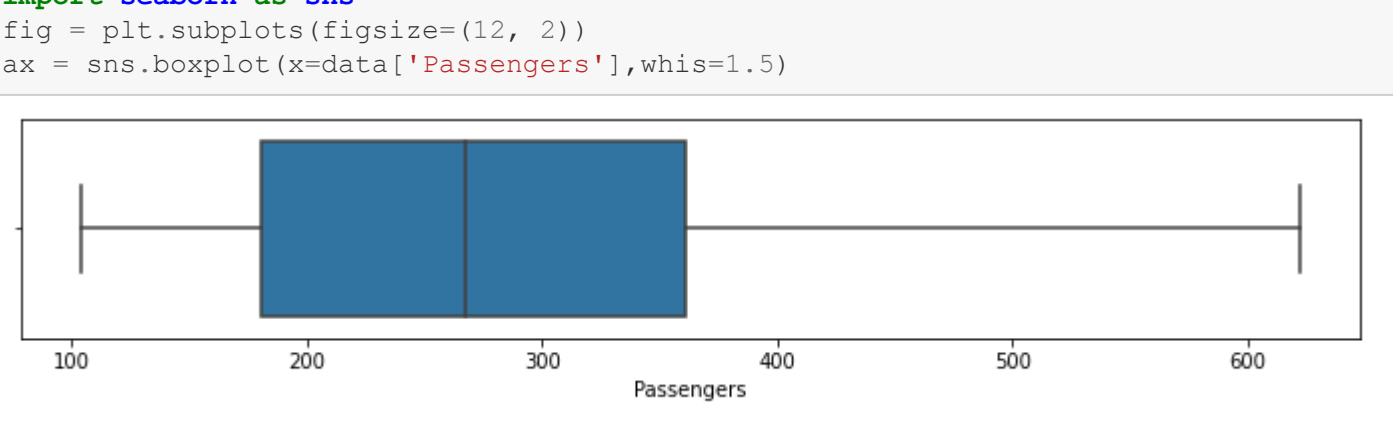
Use linear interpolation to impute missing values

In [6]:
```
data['Passengers'] = data['Passengers_Linear_Interpolation']
data.drop(columns=['Passengers_Mean_Imputation','Passengers_Linear_Interpolation'],inplace=True)
```

Outlier detection

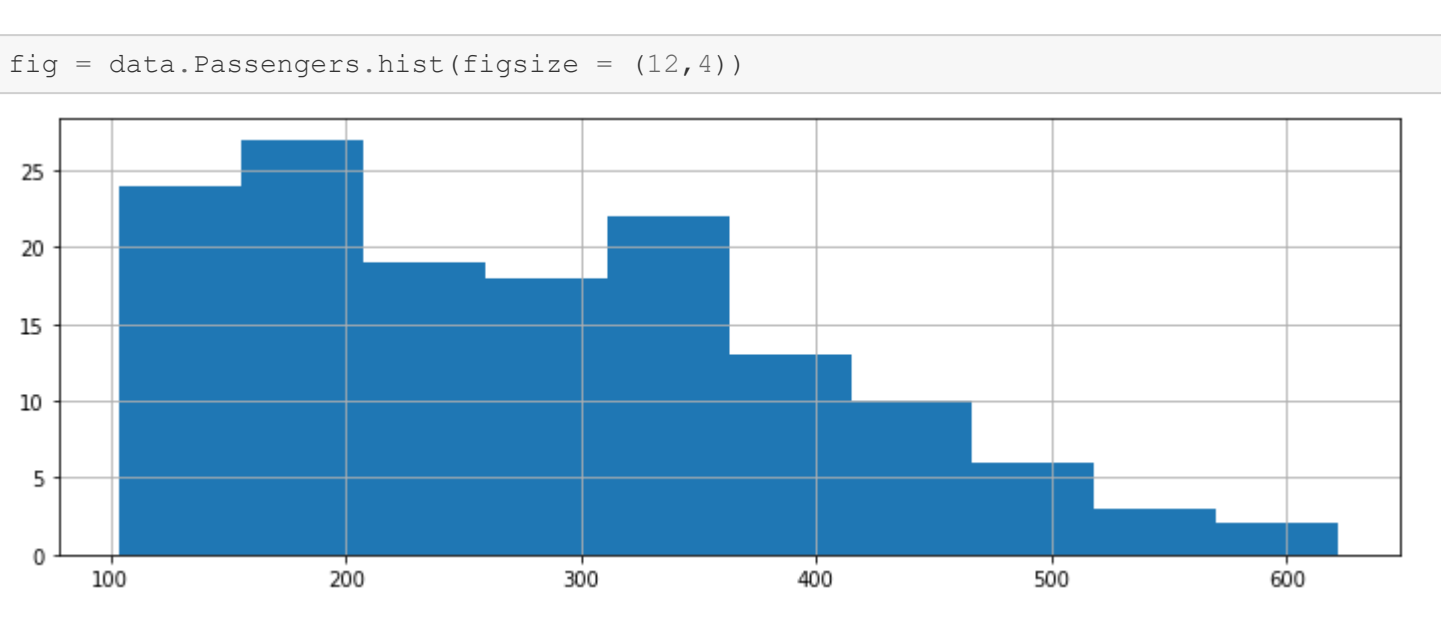
Box plot and interquartile range

In [7]:
```
import seaborn as sns
fig = plt.subplots(figsize=(12, 2))
ax = sns.boxplot(x=data['Passengers'],whis=1.5)
```



Histogram plot

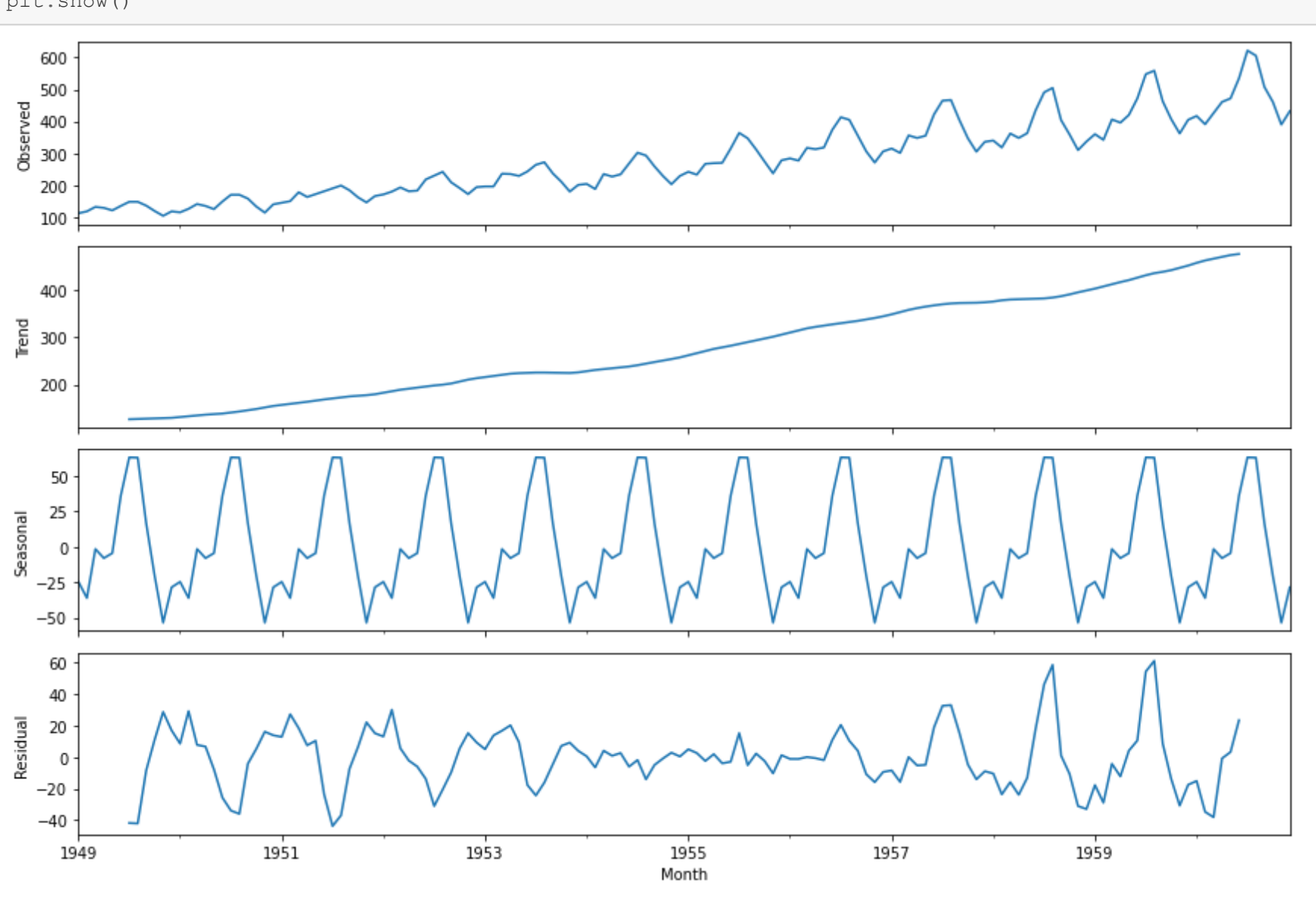
In [8]:
```
fig = data.Passengers.hist(figsize = (12,4))
```



Time series Decomposition

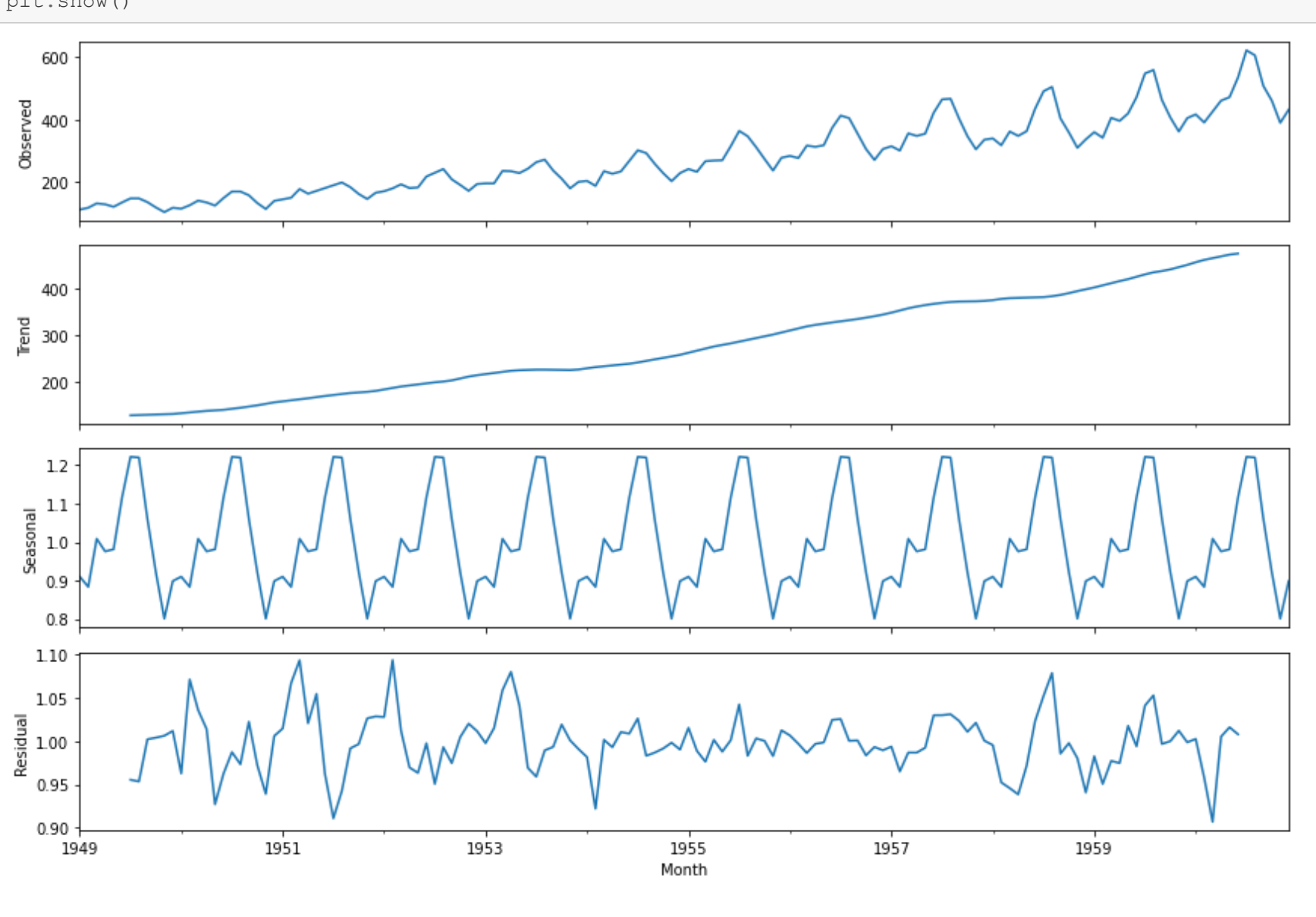
Additive seasonal decomposition

In [9]:
```
from pylab import rcParams
import statsmodels.api as sm
rcParams['figure.figsize'] = 12, 8
decomposition = sm.tsa.seasonal_decompose(data.Passengers, model='additive') # additive seasonal index
fig = decomposition.plot()
plt.show()
```



Multiplicative seasonal decomposition

In [10]:
```
decomposition = sm.tsa.seasonal_decompose(data.Passengers, model='multiplicative') # multiplicative sea
sonal index
fig = decomposition.plot()
plt.show()
```



Build and evaluate time series forecast

Split time series data into training and test set

In [11]:
```
train_len = 120
train = data[0:train_len] # first 120 months as training set
test = data[train_len:] # last 24 months as out-of-time test set
```

Simple time series methods

Naive method

In [12]:
```
y_hat_naive = test.copy()
y_hat_naive['naive_forecast'] = train['Passengers'][train_len-1]
```

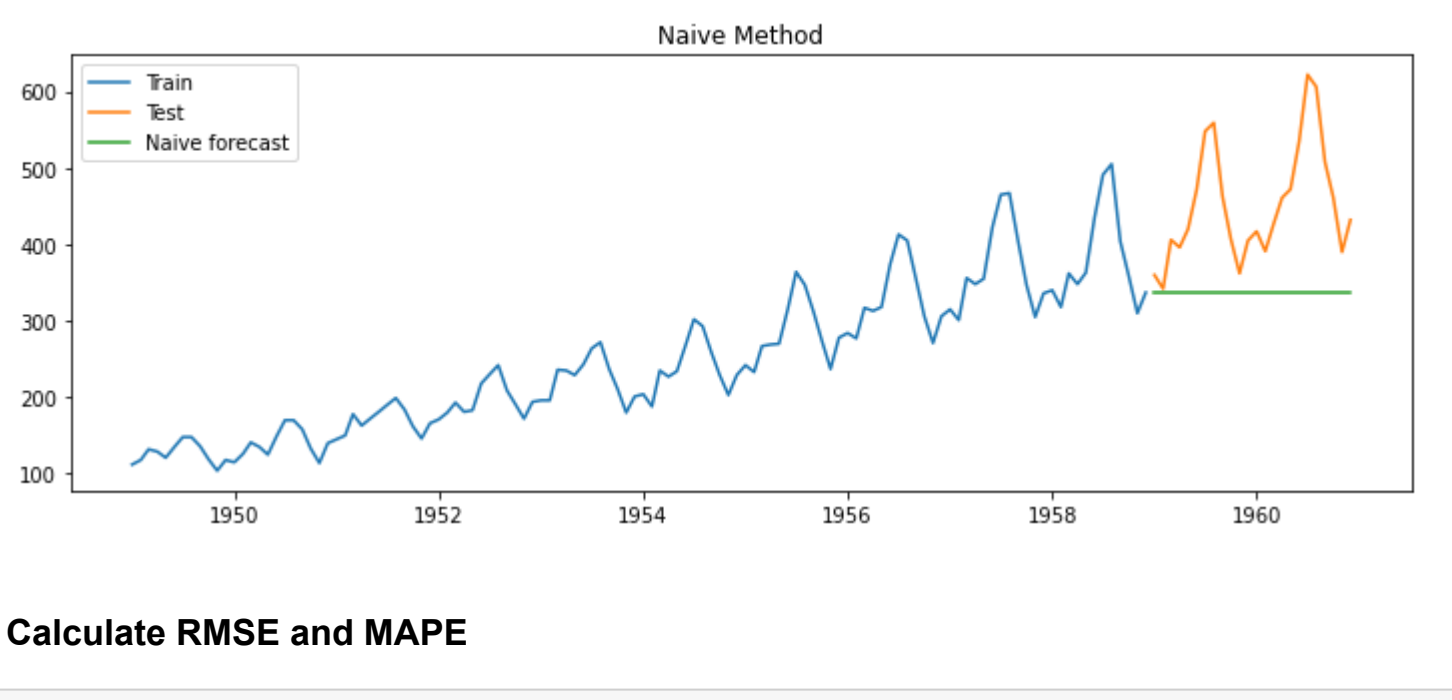
In [14]:
```
y_hat_naive
```

Out[14]:

Month	Passengers	naive_forecast
1959-01-01	360.0	337.0
1959-02-01	342.0	337.0
1959-03-01	406.0	337.0
1959-04-01	396.0	337.0
1959-05-01	420.0	337.0
1959-06-01	472.0	337.0
1959-07-01	548.0	337.0
1959-08-01	559.0	337.0
1959-09-01	463.0	337.0
1959-10-01	407.0	337.0
1959-11-01	362.0	337.0
1959-12-01	405.0	337.0
1960-01-01	417.0	337.0
1960-02-01	391.0	337.0
1960-03-01	426.0	337.0
1960-04-01	461.0	337.0
1960-05-01	472.0	337.0
1960-06-01	535.0	337.0
1960-07-01	622.0	337.0
1960-08-01	606.0	337.0
1960-09-01	508.0	337.0
1960-10-01	461.0	337.0
1960-11-01	390.0	337.0
1960-12-01	432.0	337.0

Plot train, test and forecast

In [13]:
```
plt.figure(figsize=(12,4))
plt.plot(train['Passengers'], label='Train')
plt.plot(test['Passengers'], label='Test')
plt.plot(y_hat_naive['naive_forecast'], label='Naive forecast')
plt.legend(loc='best')
plt.title('Naive Method')
plt.show()
```



Calculate RMSE and MAPE

In [14]:
```
from sklearn.metrics import mean_squared_error
rmse = np.sqrt(mean_squared_error(test['Passengers'], y_hat_naive['naive_forecast'])).round(2)
mape = np.round(np.mean(np.abs(test['Passengers']-y_hat_naive['naive_forecast'])/test['Passengers'])*10
0,2)

results = pd.DataFrame({'Method':['Naive method'], 'MAPE': [mape], 'RMSE': [rmse]})
results = results[['Method', 'RMSE', 'MAPE']]
results
```

Out[14]:

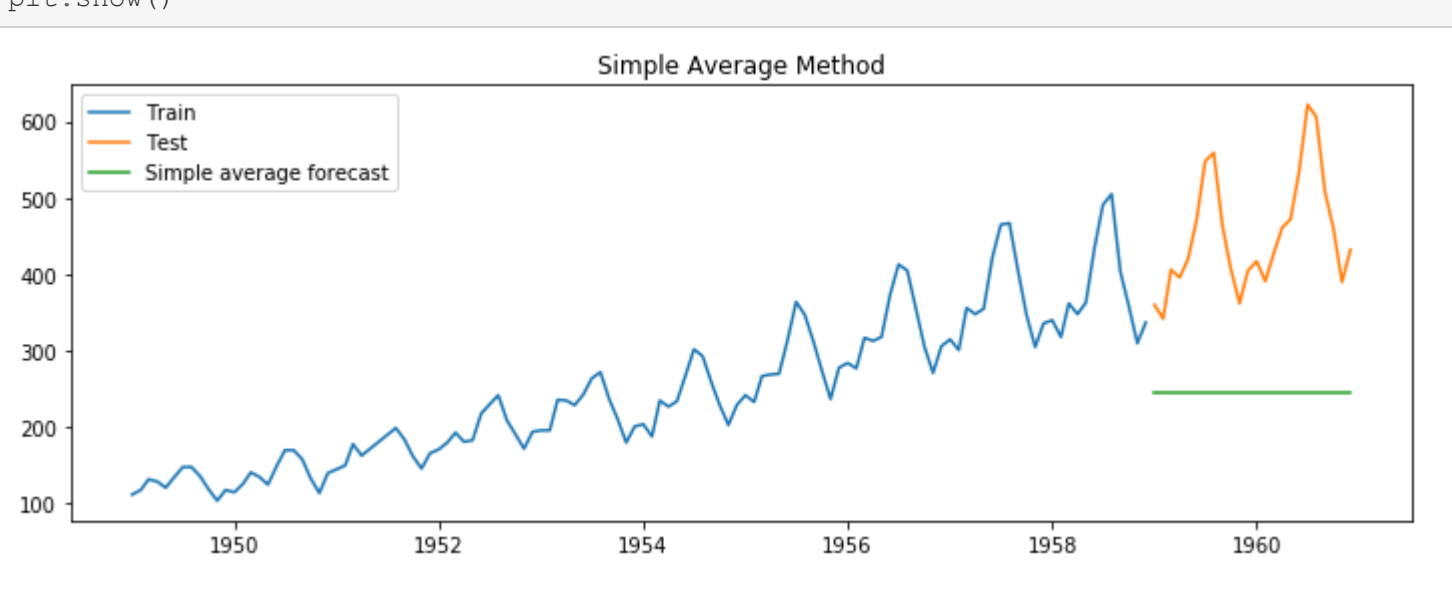
	Method	RMSE	MAPE
0	Naive method	137.51	23.63

Simple average method

In [15]:
```
y_hat_avg = test.copy()
y_hat_avg['avg_forecast'] = train['Passengers'].mean()
```

Plot train, test and forecast

In [16]:
```
plt.figure(figsize=(12,4))
plt.plot(train['Passengers'], label='Train')
plt.plot(test['Passengers'], label='Test')
plt.plot(y_hat_avg['avg_forecast'], label='Simple average forecast')
plt.legend(loc='best')
plt.title('Simple Average Method')
plt.show()
```



Calculate RMSE and MAPE

In [17]:
```
rmse = np.sqrt(mean_squared_error(test['Passengers'], y_hat_avg['avg_forecast'])).round(2)
mape = np.round(np.mean(np.abs(test['Passengers']-y_hat_avg['avg_forecast'])/test['Passengers'])*100,2)

tempResults = pd.DataFrame({'Method':['Simple average method'], 'RMSE': [rmse],'MAPE': [mape] })
results = pd.concat([results, tempResults])
results = results[['Method', 'RMSE', 'MAPE']]
results
```

Out[17]:

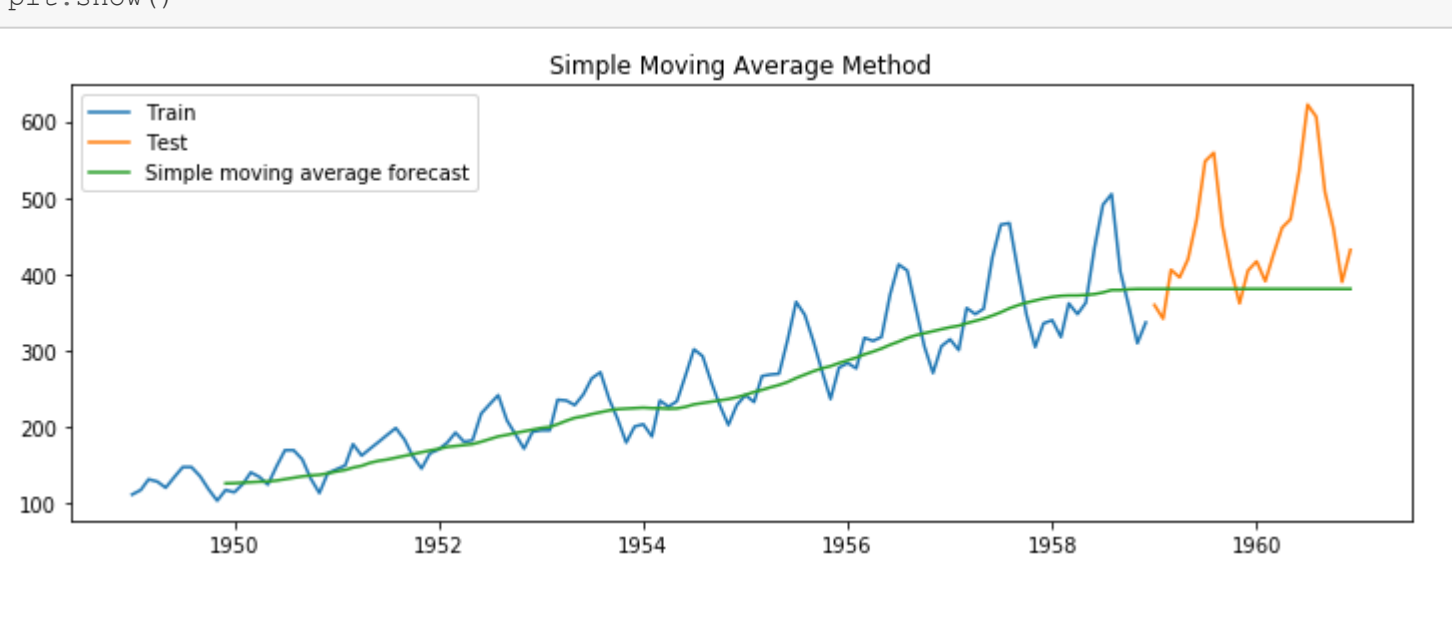
	Method	RMSE	MAPE
0	Naive method	137.51	23.63
0	Simple average method	219.69	44.28

Simple moving average method

In [18]:
```
y_hat_sma = data.copy()
ma_window = 12
y_hat_sma['sma_forecast'] = data['Passengers'].rolling(ma_window).mean()
y_hat_sma['sma_forecast'][train_len:] = y_hat_sma['sma_forecast'][train_len-1]
```

Plot train, test and forecast

In [19]:
```
plt.figure(figsize=(12,4))
plt.plot(train['Passengers'], label='Train')
plt.plot(test['Passengers'], label='Test')
plt.plot(y_hat_sma['sma_forecast'], label='Simple moving average forecast')
plt.legend(loc='best')
plt.title('Simple Moving Average Method')
plt.show()
```



Calculate RMSE and MAPE

In [20]:
```
rmse = np.sqrt(mean_squared_error(test['Passengers'], y_hat_sma['sma_forecast'][train_len:])).round(2)
mape = np.round(np.mean(np.abs(test['Passengers']-y_hat_sma['sma_forecast'][train_len:])/test['Passenge
rs'])*100,2)

tempResults = pd.DataFrame({'Method':['Simple moving average forecast'], 'RMSE': [rmse],'MAPE': [mape]
})
results = pd.concat([results, tempResults])
results = results[['Method', 'RMSE', 'MAPE']]
results
```

Out[20]:

	Method	RMSE	MAPE
0	Naive method	137.51	23.63
0	Simple average method	219.69	44.28
0	Simple moving average forecast	103.33	15.54

Exponential smoothing methods

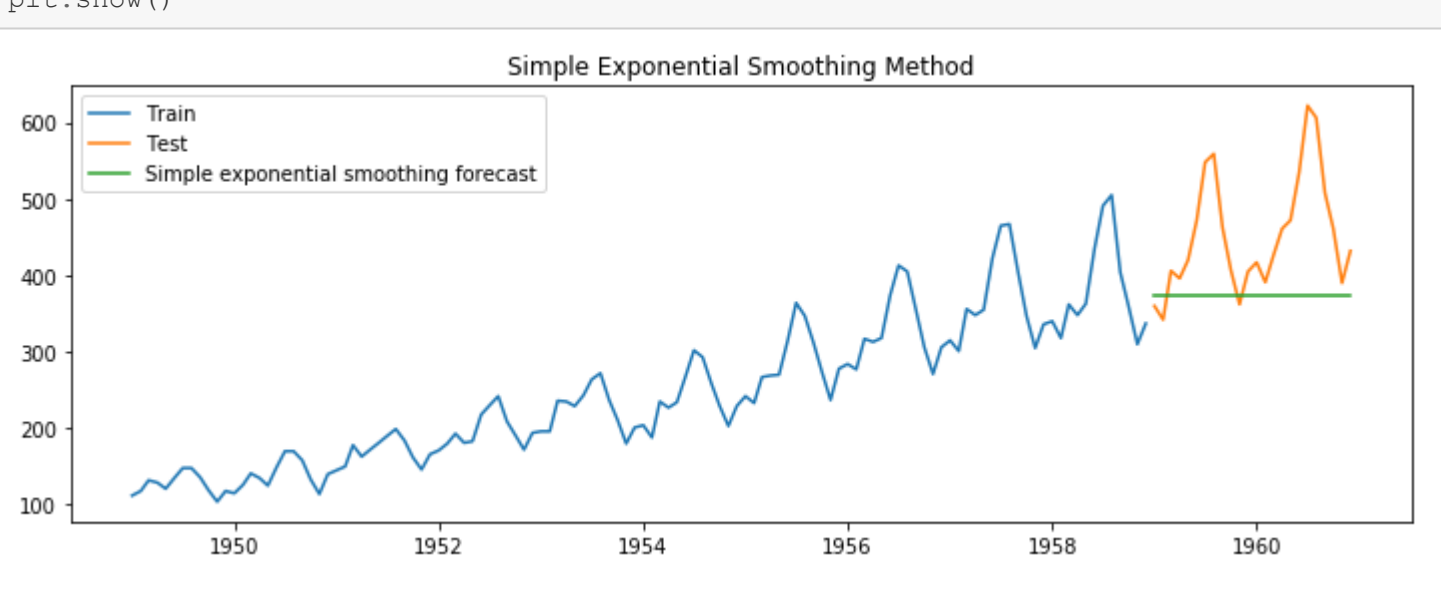
Simple exponential smoothing

In [21]:
```
from statsmodels.tsa.holtwinters import SimpleExpSmoothing
model = SimpleExpSmoothing(train['Passengers'])
model_fit = model.fit(smoothing_level=0.2,optimized=False)
model_fit.params
y_hat_ses = test.copy()
y_hat_ses['ses_forecast'] = model_fit.forecast(24)
```

```
C:\Users\25003850\Anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:165: ValueWarning: No
frequency information was provided, so inferred frequency MS will be used.
  % freq, ValueWarning)
```

Plot train, test and forecast

In [22]:
```
plt.figure(figsize=(12,4))
plt.plot(train['Passengers'], label='Train')
plt.plot(test['Passengers'], label='Test')
plt.plot(y_hat_ses['ses_forecast'], label='Simple exponential smoothing forecast')
plt.legend(loc='best')
plt.title('Simple Exponential Smoothing Method')
plt.show()
```



Calculate RMSE and MAPE

In [23]:
```
rmse = np.sqrt(mean_squared_error(test['Passengers'], y_hat_ses['ses_forecast'])).round(2)
mape = np.round(np.mean(np.abs(test['Passengers']-y_hat_ses['ses_forecast'])/test['Passengers'])*100,2)

tempResults = pd.DataFrame({'Method':['Simple exponential smoothing forecast'], 'RMSE': [rmse],'MAPE':
[mape] })
results = pd.concat([results, tempResults])
results
```

Out[23]:

	Method	RMSE	MAPE
0	Naive method	137.51	23.63
0	Simple average method	219.69	44.28
0	Simple moving average forecast	103.33	15.54
0	Simple exponential smoothing forecast	107.65	16.49

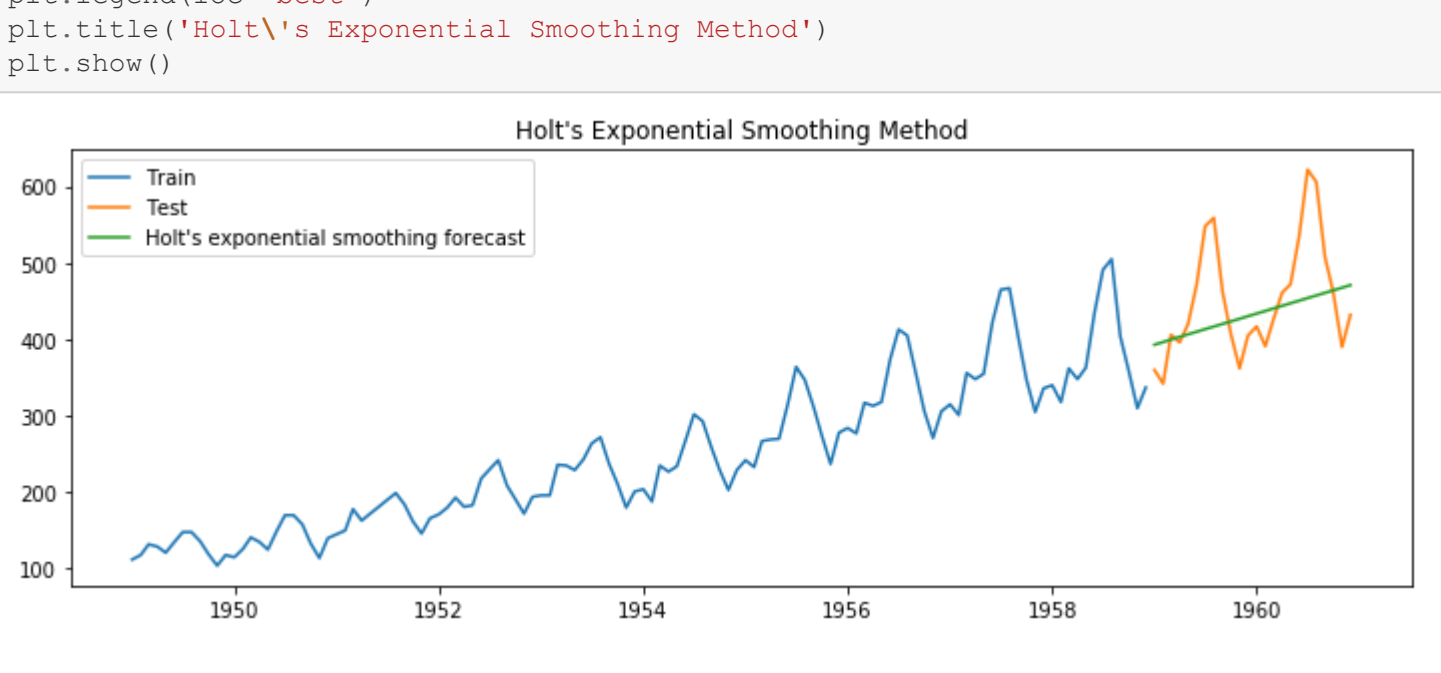
Holt's method with trend

In [24]:
```
from statsmodels.tsa.holtwinters import ExponentialSmoothing
model = ExponentialSmoothing(np.asarray(train['Passengers']) ,seasonal_periods=12 ,trend='additive', se
asonal=None)
model_fit = model.fit(smoothing_level=0.2, smoothing_slope=0.01, optimized=False)
print(model_fit.params)
y_hat_holt = test.copy()
y_hat_holt['holt_forecast'] = model_fit.forecast(len(test))
```

```
{'smoothing_level': 0.2, 'smoothing_slope': 0.01, 'smoothing_seasonal': None, 'damping_slope': nan,
'initial_level': 112.0, 'initial_slope': 6.0, 'initial_seasons': array([], dtype=float64), 'use_boxco
x': False, 'lamda': None, 'remove_bias': False}
```

Plot train, test and forecast

In [25]:
```
plt.figure(figsize=(12,4))
plt.plot( train['Passengers'], label='Train')
plt.plot(test['Passengers'], label='Test')
plt.plot(y_hat_holt['holt_forecast'], label='Holt\'s exponential smoothing forecast')
plt.legend(loc='best')
plt.title('Holt\'s Exponential Smoothing Method')
plt.show()
```



Calculate RSME and MAPE

In [26]:
```
rmse = np.sqrt(mean_squared_error(test['Passengers'], y_hat_holt['holt_forecast'])).round(2)
mape = np.round(np.mean(np.abs(test['Passengers']-y_hat_holt['holt_forecast'])/test['Passengers'])*100,
2)

tempResults = pd.DataFrame({'Method':['Holt\'s exponential smoothing method'], 'RMSE': [rmse],'MAPE': [
mape] })
results = pd.concat([results, tempResults])
results = results[['Method', 'RMSE', 'MAPE']]
results
```

Out[26]:

	Method	RMSE	MAPE
0	Naive method	137.51	23.63
0	Simple average method	219.69	44.28
0	Simple moving average forecast	103.33	15.54
0	Simple exponential smoothing forecast	107.65	16.49
0	Holt's exponential smoothing method	71.94	11.11

Holt Winters' additive method with trend and seasonality

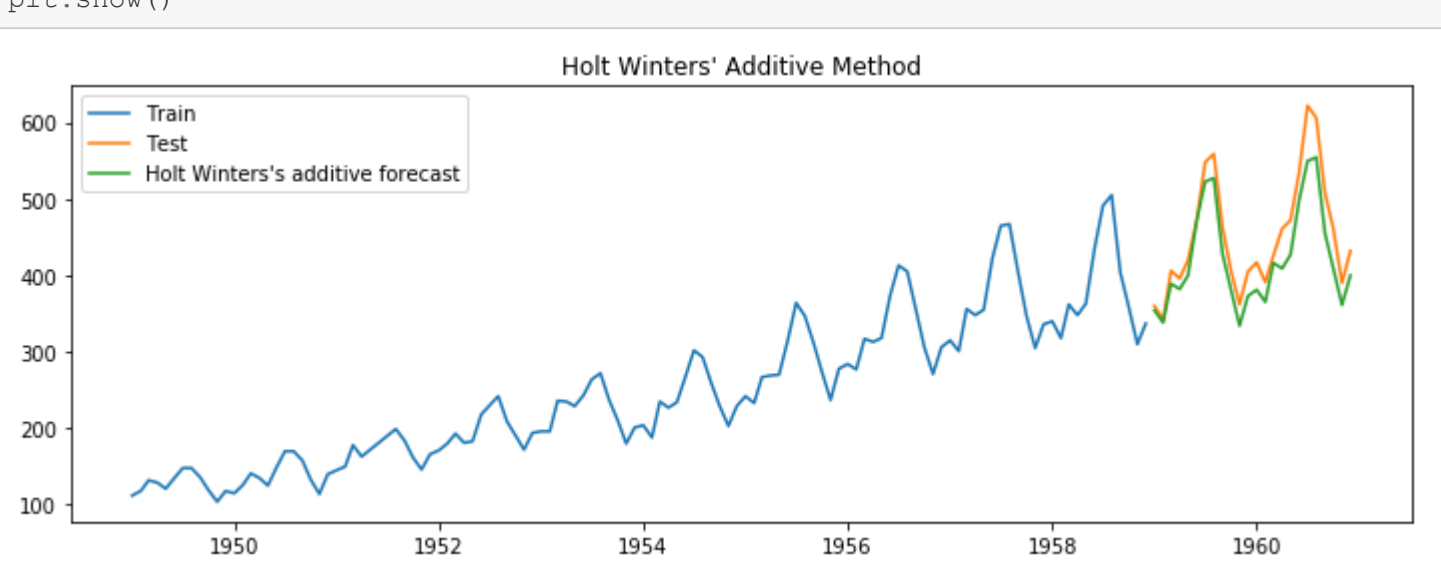
In [27]:
```
y_hat_hwa = test.copy()
model = ExponentialSmoothing(np.asarray(train['Passengers']) ,seasonal_periods=12 ,trend='add', seasona
l='add')
model_fit = model.fit(optimized=True)
print(model_fit.params)
y_hat_hwa['hw_forecast'] = model_fit.forecast(24)
```

```
{'smoothing_level': 0.26315789314098276, 'smoothing_slope': 0.051629645024921914, 'smoothing_seasona
l': 0.7368421036672985, 'damping_slope': nan, 'initial_level': 211.5509857554829, 'initial_slope': 1.
7392015465813395, 'initial_seasons': array([-100.18118104,  -94.20464473,  -80.29175401,  -83.4175839
1,
        -91.45908167,  -77.45094418,  -64.48324484,  -64.57340828,
        -76.66644581,  -93.73584652, -108.67611385,  -94.50947874]), 'use_boxcox': False, 'lamda': No
ne, 'remove_bias': False}
```

```
C:\Users\25003850\Anaconda3\lib\site-packages\statsmodels\tsa\holtwinters.py:712: ConvergenceWarning:
Optimization failed to converge. Check mle_retvals.
  ConvergenceWarning)
```

Plot train, test and forecast

In [28]:
```
plt.figure(figsize=(12,4))
plt.plot( train['Passengers'], label='Train')
plt.plot(test['Passengers'], label='Test')
plt.plot(y_hat_hwa['hw_forecast'], label='Holt Winters\'s additive forecast')
plt.legend(loc='best')
plt.title('Holt Winters\' Additive Method')
plt.show()
```



Calculate RMSE and MAPE

In [29]:
```
rmse = np.sqrt(mean_squared_error(test['Passengers'], y_hat_hwa['hw_forecast'])).round(2)
mape = np.round(np.mean(np.abs(test['Passengers']-y_hat_hwa['hw_forecast'])/test['Passengers'])*100,2)

tempResults = pd.DataFrame({'Method':['Holt Winters\' additive method'], 'RMSE': [rmse],'MAPE': [mape]
})
results = pd.concat([results, tempResults])
results = results[['Method', 'RMSE', 'MAPE']]
results
```

Out[29]:

	Method	RMSE	MAPE
0	Naive method	137.51	23.63
0	Simple average method	219.69	44.28
0	Simple moving average forecast	103.33	15.54
0	Simple exponential smoothing forecast	107.65	16.49
0	Holt's exponential smoothing method	71.94	11.11
0	Holt Winters' additive method	35.10	6.53

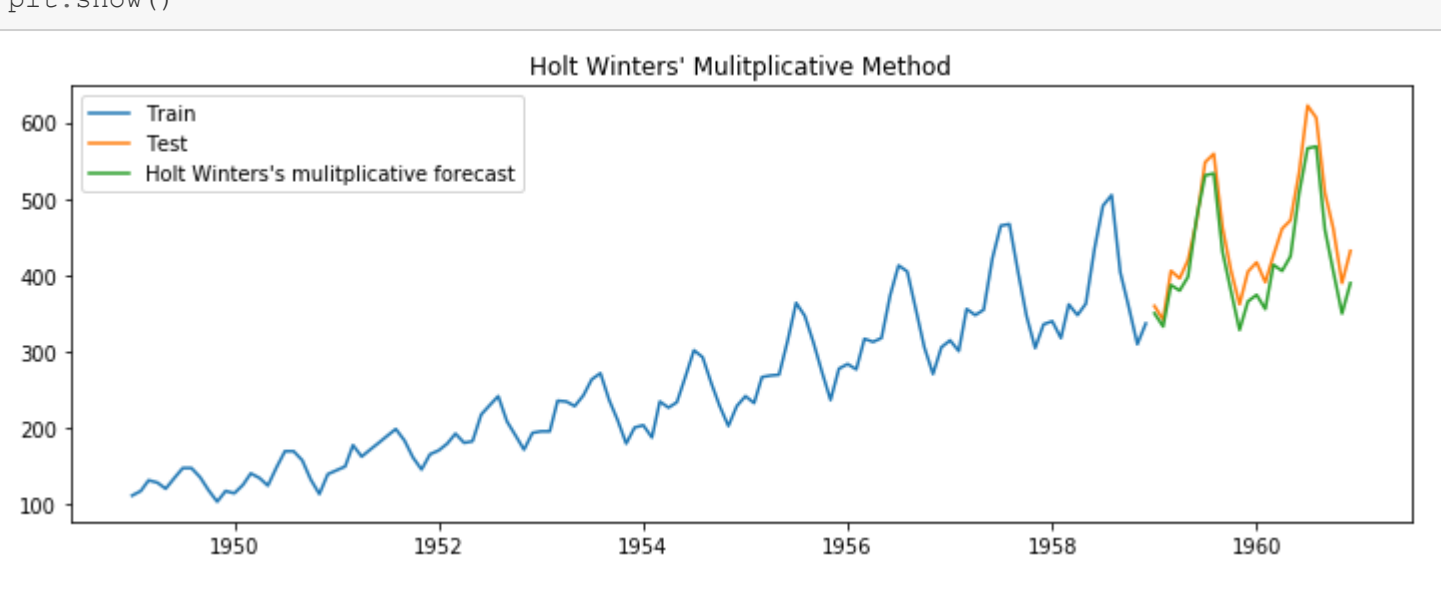
Holt Winter's multiplicative method with trend and seasonality

In [30]:
```
y_hat_hwm = test.copy()
model = ExponentialSmoothing(np.asarray(train['Passengers']) ,seasonal_periods=12 ,trend='add', seasona
l='mul')
model_fit = model.fit(optimized=True)
print(model_fit.params)
y_hat_hwm['hw_forecast'] = model_fit.forecast(24)
```

```
{'smoothing_level': 0.38484064840698556, 'smoothing_slope': 0.035386453247783, 'smoothing_seasonal':
0.6151593186319729, 'damping_slope': nan, 'initial_level': 212.39830870038878, 'initial_slope': 1.105
7967107993691, 'initial_seasons': array([0.51755725, 0.54595018, 0.60261532, 0.56966257, 0.52017629,
       0.5722879 , 0.62749981, 0.62520353, 0.56963344, 0.49468435,
       0.43937027, 0.51362925]), 'use_boxcox': False, 'lamda': None, 'remove_bias': False}
```

Plot train, test and forecast

In [31]:
```
plt.figure(figsize=(12,4))
plt.plot( train['Passengers'], label='Train')
plt.plot(test['Passengers'], label='Test')
plt.plot(y_hat_hwm['hw_forecast'], label='Holt Winters\'s mulitplicative forecast')
plt.legend(loc='best')
plt.title('Holt Winters\' Mulitplicative Method')
plt.show()
```



Calculate RMSE and MAPE

In [32]:
```
rmse = np.sqrt(mean_squared_error(test['Passengers'], y_hat_hwm['hw_forecast'])).round(2)
mape = np.round(np.mean(np.abs(test['Passengers']-y_hat_hwm['hw_forecast'])/test['Passengers'])*100,2)

tempResults = pd.DataFrame({'Method':['Holt Winters\' multiplicative method'], 'RMSE': [rmse],'MAPE': [
mape] })
results = pd.concat([results, tempResults])
results = results[['Method', 'RMSE', 'MAPE']]
results
```

Out[32]:

	Method	RMSE	MAPE
0	Naive method	137.51	23.63
0	Simple average method	219.69	44.28
0	Simple moving average forecast	103.33	15.54
0	Simple exponential smoothing forecast	107.65	16.49
0	Holt's exponential smoothing method	71.94	11.11
0	Holt Winters' additive method	35.10	6.53
0	Holt Winters' multiplicative method	34.83	6.91