auto-regressive-methods

December 29, 2023

[178]:
```
import pandas as pd
import numpy as np
import seaborn as sns

import matplotlib.pyplot as plt
from statsmodels.tsa.seasonal import seasonal_decompose
from sklearn.metrics import mean_squared_error
```

[179]:
```
data = pd.read_csv('/kaggle/input/air-passengers/AirPassengers.csv')
data.head()
```

[179]:

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

[180]:
```
data = data.rename(columns={"#Passengers": "Passengers"}, inplace=False)
data.head()
```

[180]:

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

[181]:
```
data.columns = ['Month','Passengers']
data['Month'] = pd.to_datetime(data['Month'], format='%Y-%m')
data = data.set_index('Month')
data.head()
```

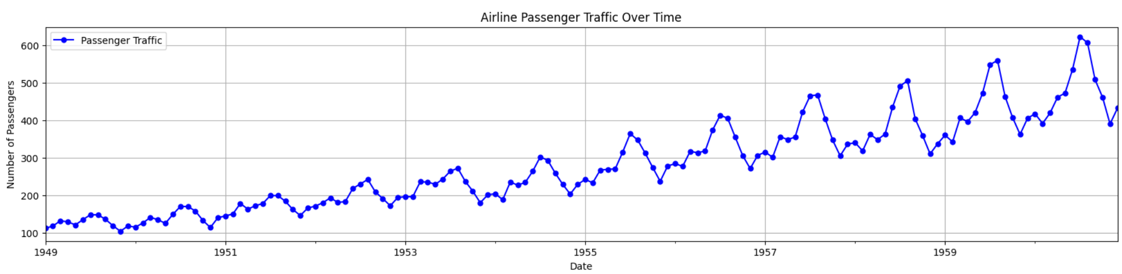
[181]:

	Passengers
Month	
1949-01-01	112
1949-02-01	118

```
1949-03-01          132
1949-04-01          129
1949-05-01          121
```

1 Plot time series data

[182]:
```
data.plot(y='Passengers', figsize=(20, 4), color='blue', linestyle='-',␣
↪marker='o', markersize=5, label='Passenger Traffic')
plt.grid(True)
plt.legend(loc='best')
plt.title('Airline Passenger Traffic Over Time')
plt.xlabel('Date')
plt.ylabel('Number of Passengers')
plt.show(block=False)
```

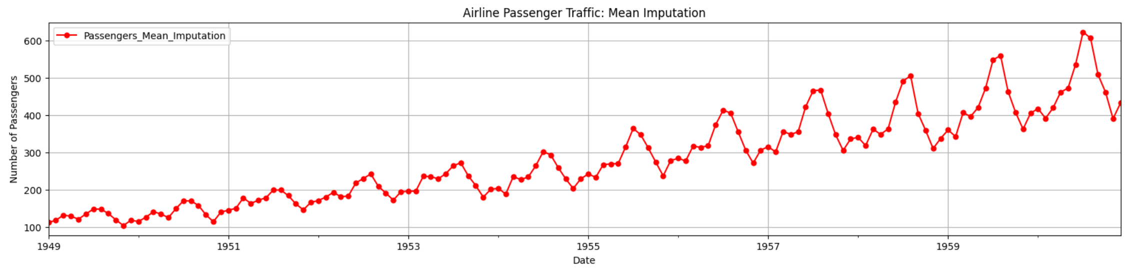


2 Missing value treatment

2.1 Mean imputation

[183]:
```
data['Passengers_Mean_Imputation'] = data['Passengers'].
↪fillna(data['Passengers'].mean())
```

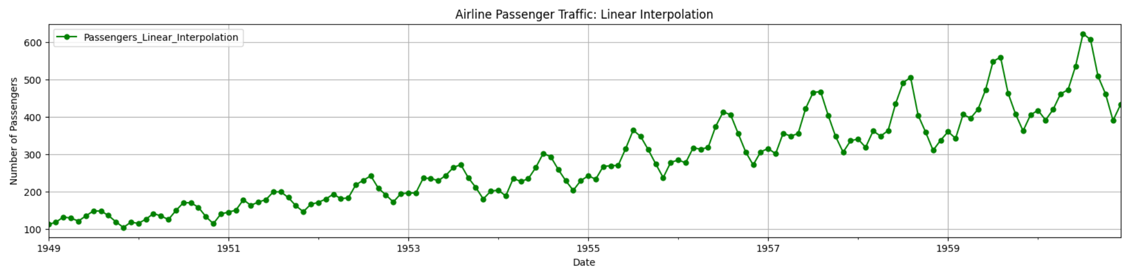
[184]:
```
data[['Passengers_Mean_Imputation']].plot(figsize=(20, 4), grid=True,␣
↪legend=True, color='red', linestyle='-', marker='o', markersize=5)
plt.title('Airline Passenger Traffic: Mean Imputation')
plt.xlabel('Date')
plt.ylabel('Number of Passengers')
plt.show(block=False)
```



2.2 Linear interpolation

[185]:
```python
data['Passengers_Linear_Interpolation'] = data['Passengers'].
    ↪interpolate(method='linear')
```

[186]:
```python
data[['Passengers_Linear_Interpolation']].plot(figsize=(20, 4), grid=True,␣
    ↪legend=True, color='green', linestyle='-', marker='o', markersize=5)
plt.title('Airline Passenger Traffic: Linear Interpolation')
plt.xlabel('Date')
plt.ylabel('Number of Passengers')
plt.show(block=False)
```



[187]:
```python
data.head()
```

[187]:
```
            Passengers  Passengers_Mean_Imputation  \
Month
1949-01-01         112                         112
1949-02-01         118                         118
1949-03-01         132                         132
1949-04-01         129                         129
1949-05-01         121                         121

            Passengers_Linear_Interpolation
Month
1949-01-01                              112
```

```
1949-02-01                                 118
1949-03-01                                 132
1949-04-01                                 129
1949-05-01                                 121
```

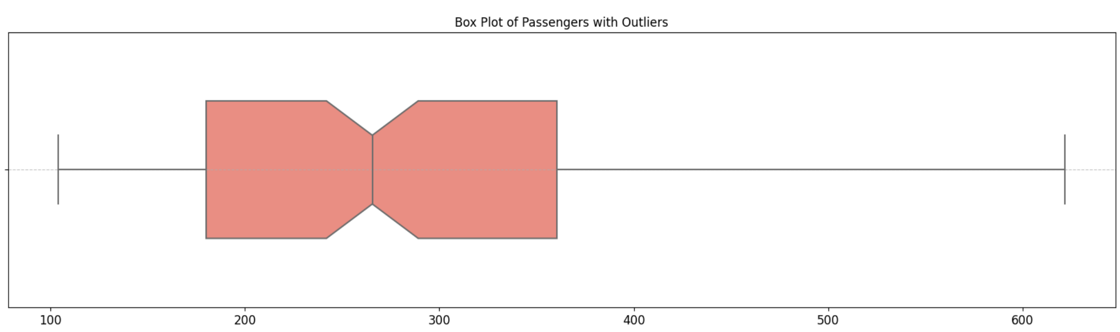
[188]:
```
data['Passengers'] = data['Passengers_Linear_Interpolation']
data.
 ↪drop(columns=['Passengers_Mean_Imputation','Passengers_Linear_Interpolation'],inplace=True)
data.head()
```

[188]:
```
            Passengers
Month
1949-01-01         112
1949-02-01         118
1949-03-01         132
1949-04-01         129
1949-05-01         121
```

3 Outlier detection

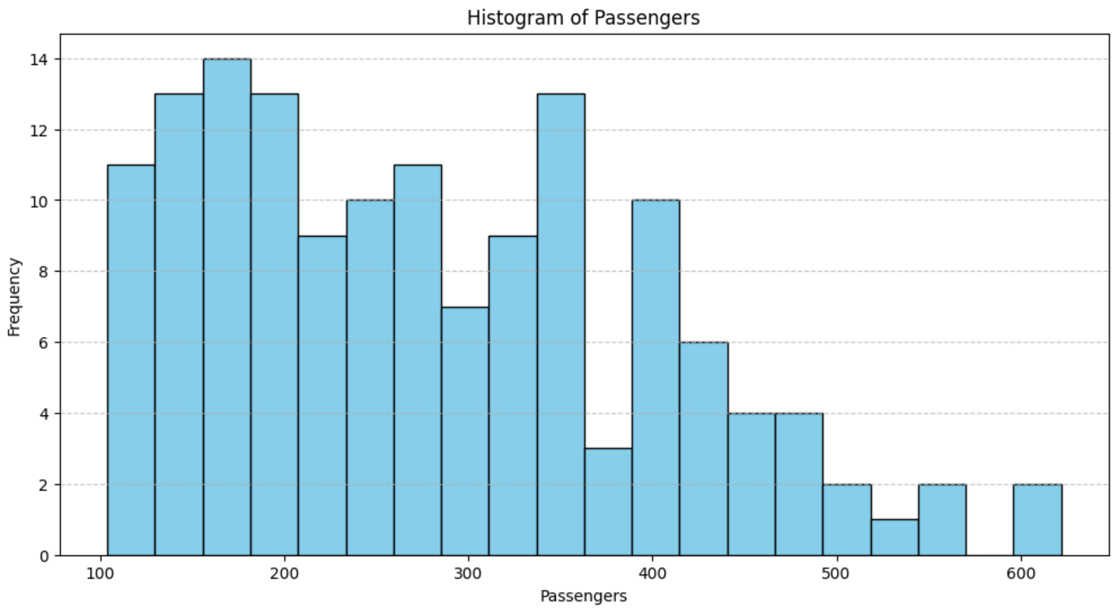
3.1 Box plot and interquartile range

[189]:
```
import seaborn as sns
plt.figure(figsize=(20, 5))

sns.boxplot(x=data['Passengers'], color='salmon', width=0.5, notch=True)

plt.title('Box Plot of Passengers with Outliers')
plt.xlabel('Passengers')
plt.xticks(fontsize=12)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



3.2 Histogram plot

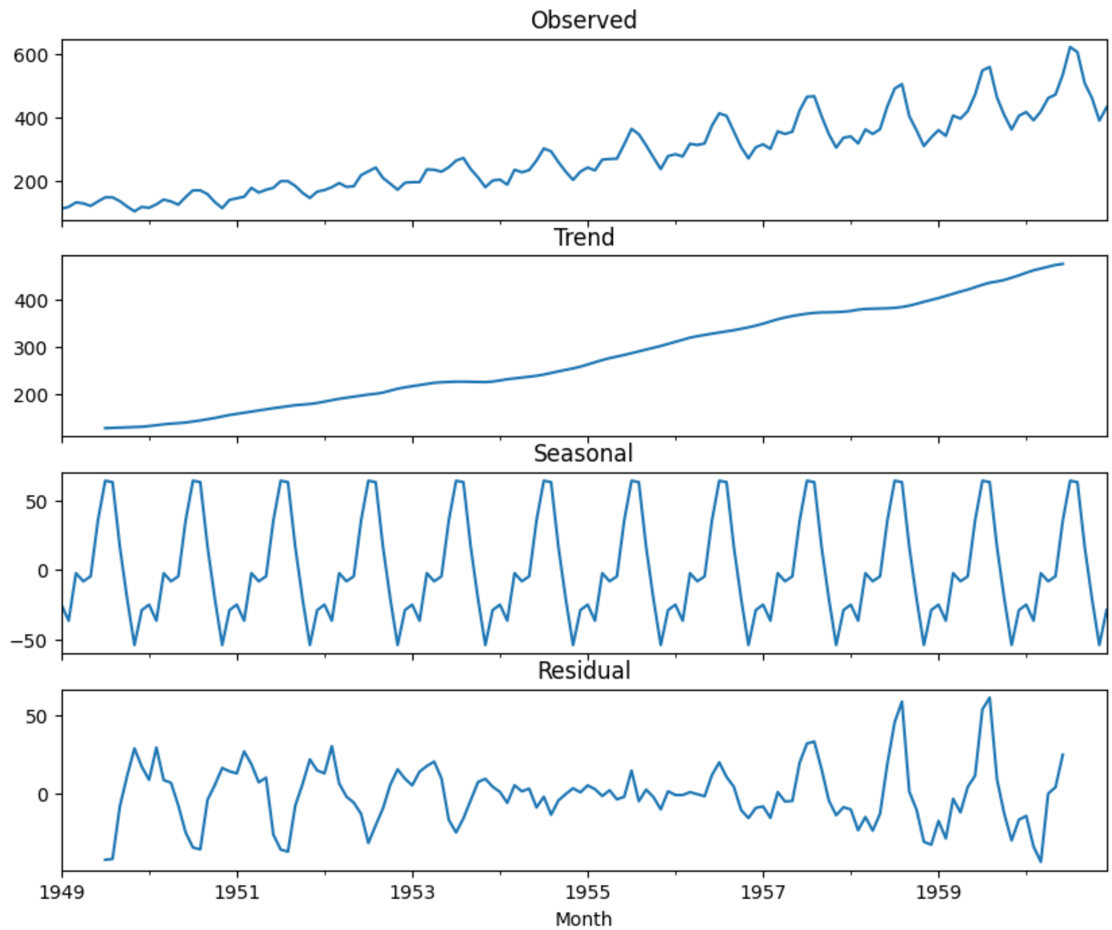
[190]:
```
plt.figure(figsize=(12, 6))
plt.hist(data['Passengers'], bins=20, color='skyblue', edgecolor='black')

plt.title('Histogram of Passengers')
plt.xlabel('Passengers')
plt.ylabel('Frequency')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



4 Time series Decomposition

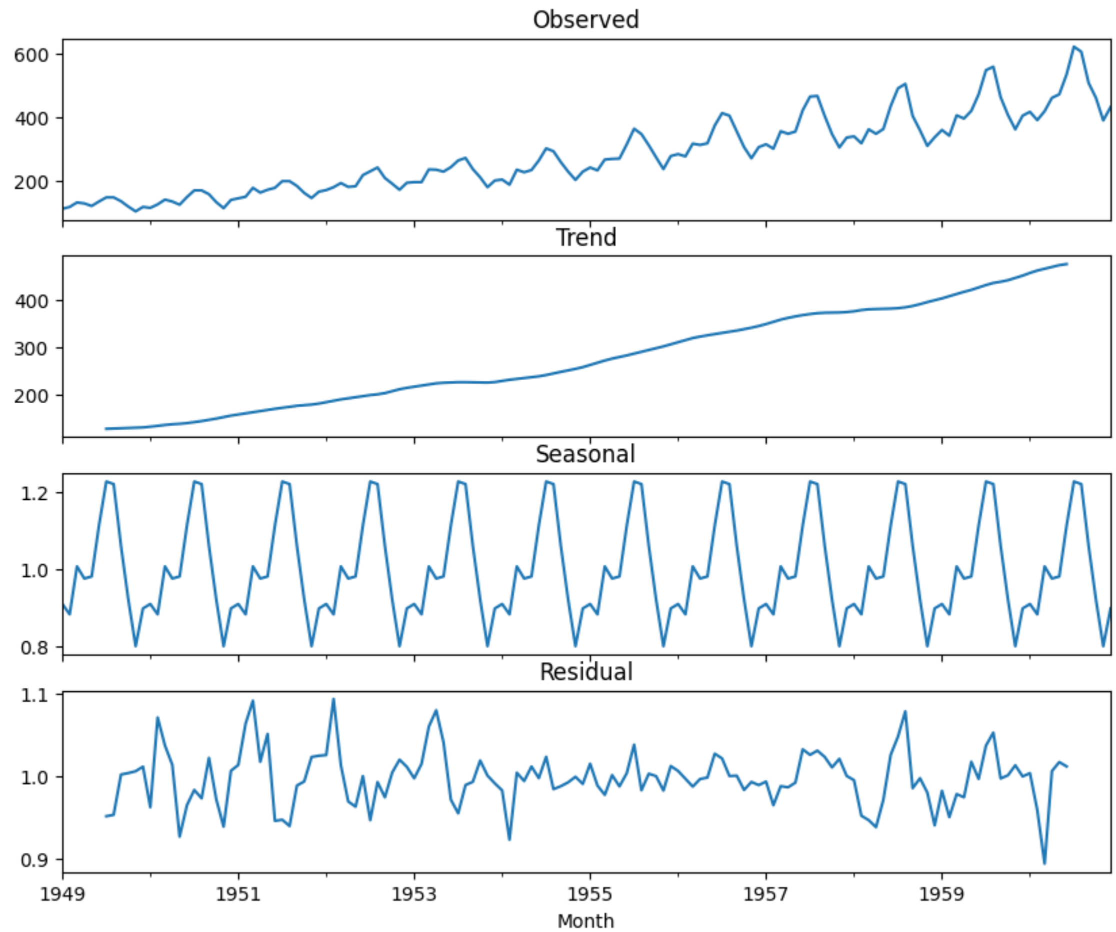
4.1 Additive seasonal decomposition

[191]:
```
from statsmodels.tsa.seasonal import seasonal_decompose

# Perform decomposition
result = seasonal_decompose(data['Passengers'], model='additive')
fig, (ax1, ax2, ax3, ax4) = plt.subplots(4, 1, figsize=(10, 8), sharex=True)

result.observed.plot(ax=ax1, title='Observed')
result.trend.plot(ax=ax2, title='Trend')
result.seasonal.plot(ax=ax3, title='Seasonal')
result.resid.plot(ax=ax4, title='Residual')

plt.show()
```



4.2 Multiplicative seasonal decomposition

[192]:
```
result = seasonal_decompose(data['Passengers'], model='multiplicative')
fig, (ax1, ax2, ax3, ax4) = plt.subplots(4, 1, figsize=(10, 8), sharex=True)

result.observed.plot(ax=ax1, title='Observed')
result.trend.plot(ax=ax2, title='Trend')
result.seasonal.plot(ax=ax3, title='Seasonal')
result.resid.plot(ax=ax4, title='Residual')

plt.show()
```



5 Build and evaluate time series forecast

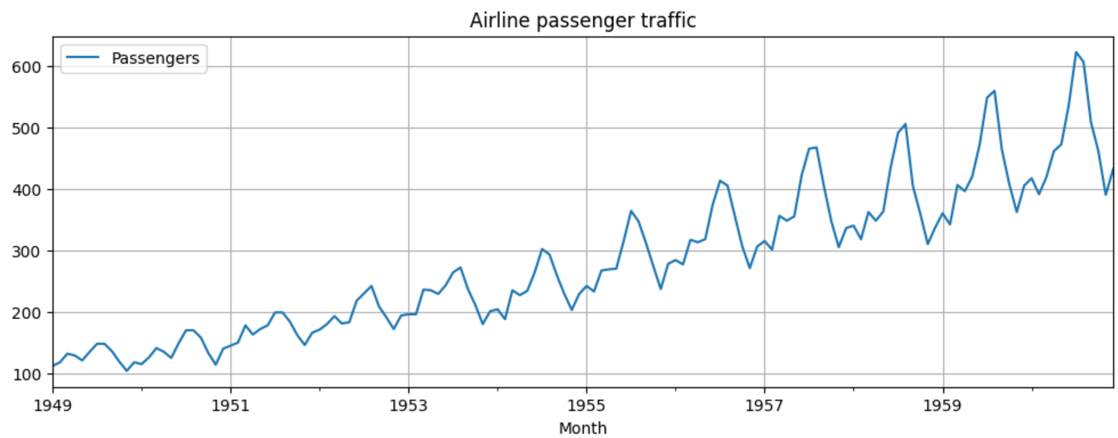
5.1 Split time series data into training and test set

[193]:
```
train_len = 120
train = data[:train_len]  # first 120 months as the training set
test = data[train_len:]  # last 24 months as the out-of-time test set
```

6 Auto Regressive methods

6.1 Stationarity vs non-stationary time series

[194]:
```
data['Passengers'].plot(figsize=(12, 4))
plt.grid()
plt.legend(loc='best')
plt.title('Airline passenger traffic')
plt.show(block=False)
```



6.2 Augmented Dickey-Fuller (ADF) test

[195]:
```
from statsmodels.tsa.stattools import adfuller
adf_test = adfuller(data['Passengers'])

print('ADF Statistic: %f' % adf_test[0])
print('Critical Values @ 0.05: %.2f' % adf_test[4]['5%'])
print('p-value: %f' % adf_test[1])
```

```
ADF Statistic: 0.815369
Critical Values @ 0.05: -2.88
p-value: 0.991880
```

6.3 Kwiatkowski-Phillips-Schmidt-Shin (KPSS) test

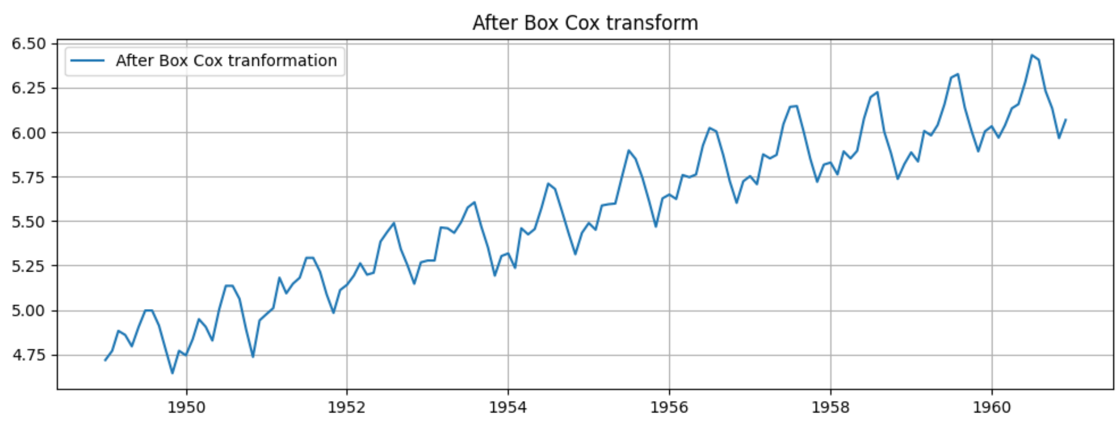
[196]:
```
from statsmodels.tsa.stattools import kpss
kpss_test = kpss(data['Passengers'])

print('KPSS Statistic: %f' % kpss_test[0])
print('Critical Values @ 0.05: %.2f' % kpss_test[3]['5%'])
print('p-value: %f' % kpss_test[1])
```

```
KPSS Statistic: 1.651312
Critical Values @ 0.05: 0.46
p-value: 0.010000
```

```
/tmp/ipykernel_43/3602609379.py:2: InterpolationWarning: The test statistic is
outside of the range of p-values available in the
look-up table. The actual p-value is smaller than the p-value returned.

  kpss_test = kpss(data['Passengers'])
```

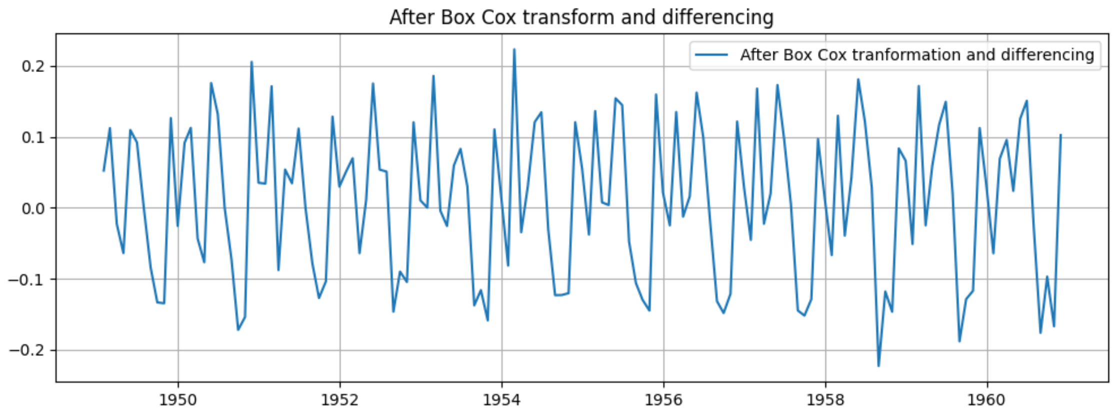
6.4 Box Cox transformation to make variance constant

[197]:
```
from scipy.stats import boxcox
data_boxcox = pd.Series(boxcox(data['Passengers'], lmbda=0), index = data.index)

plt.figure(figsize=(12,4))
plt.grid()
plt.plot(data_boxcox, label='After Box Cox tranformation')
plt.legend(loc='best')
plt.title('After Box Cox transform')
plt.show()
```



6.5 Differencing to remove trend

[198]:
```
data_boxcox_diff = pd.Series(data_boxcox - data_boxcox.shift(), data.index)
plt.figure(figsize=(12,4))
plt.grid()
plt.plot(data_boxcox_diff, label='After Box Cox tranformation and differencing')
plt.legend(loc='best')
plt.title('After Box Cox transform and differencing')
plt.show()
```



[199]:
```
data_boxcox_diff.dropna(inplace=True)
data_boxcox_diff.tail()
```

[199]:
```
Month
1960-08-01   -0.026060
1960-09-01   -0.176399
1960-10-01   -0.097083
1960-11-01   -0.167251
1960-12-01    0.102279
dtype: float64
```

6.6 Augmented Dickey-Fuller (ADF) test

[200]:
```
adf_test = adfuller(data_boxcox_diff)

print('ADF Statistic: %f' % adf_test[0])
print('Critical Values @ 0.05: %.2f' % adf_test[4]['5%'])
print('p-value: %f' % adf_test[1])
```

```
ADF Statistic: -2.717131
Critical Values @ 0.05: -2.88
p-value: 0.071121
```

6.7 Kwiatkowski-Phillips-Schmidt-Shin (KPSS) test

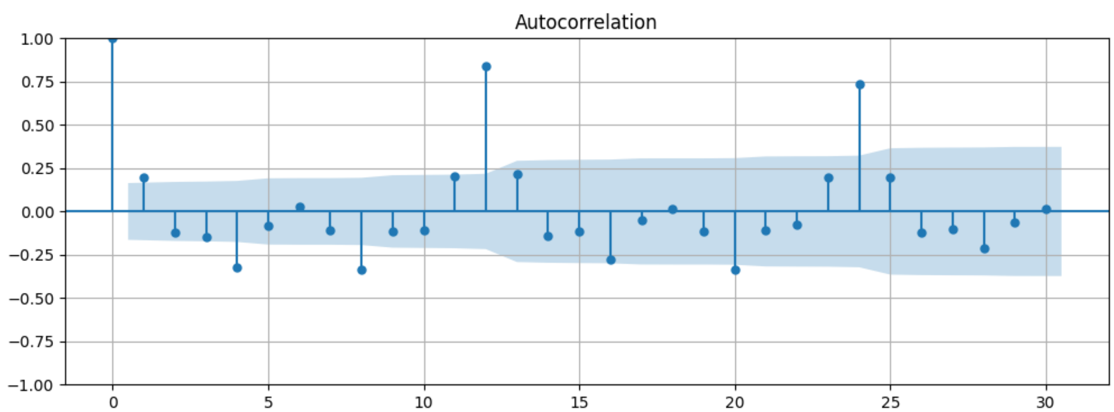
[201]:
```
kpss_test = kpss(data_boxcox_diff)

print('KPSS Statistic: %f' % kpss_test[0])
print('Critical Values @ 0.05: %.2f' % kpss_test[3]['5%'])
print('p-value: %f' % kpss_test[1])
```

```
KPSS Statistic: 0.038304
Critical Values @ 0.05: 0.46
p-value: 0.100000
```

```
/tmp/ipykernel_43/3639712988.py:1: InterpolationWarning: The test statistic is
outside of the range of p-values available in the
look-up table. The actual p-value is greater than the p-value returned.

  kpss_test = kpss(data_boxcox_diff)
```

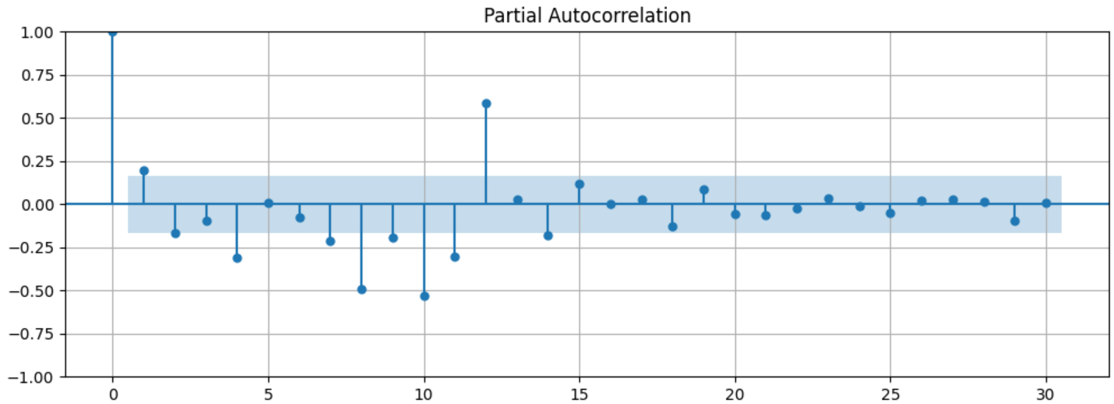
6.8 Autocorrelation function (ACF)

[202]:
```
from statsmodels.graphics.tsaplots import plot_acf
plt.figure(figsize=(12,4))
plt.grid()
plot_acf(data_boxcox_diff, ax=plt.gca(), lags = 30)
plt.show()
```



6.9 Partial autocorrelation function (PACF)

[203]:
```
from statsmodels.graphics.tsaplots import plot_pacf
plt.figure(figsize=(12,4))
plt.grid()
plot_pacf(data_boxcox_diff, ax=plt.gca(), lags = 30)
plt.show()
```



[204]:
```
train_data_boxcox = data_boxcox[:train_len]
test_data_boxcox = data_boxcox[train_len:]
train_data_boxcox_diff = data_boxcox_diff[:train_len-1]
test_data_boxcox_diff = data_boxcox_diff[train_len-1:]
```

[205]:
```
train_data_boxcox_diff[:10]
```

[205]:
```
Month
1949-02-01    0.052186
1949-03-01    0.112117
1949-04-01   -0.022990
1949-05-01   -0.064022
1949-06-01    0.109484
1949-07-01    0.091937
1949-08-01    0.000000
1949-09-01   -0.084557
1949-10-01   -0.133531
1949-11-01   -0.134733
dtype: float64
```

7 Auto regression method (AR)

[206]:
```
from statsmodels.tsa.arima.model import ARIMA
model = ARIMA(train_data_boxcox_diff, order=(1, 0, 0))
model_fit = model.fit()
print(model_fit.params)
```

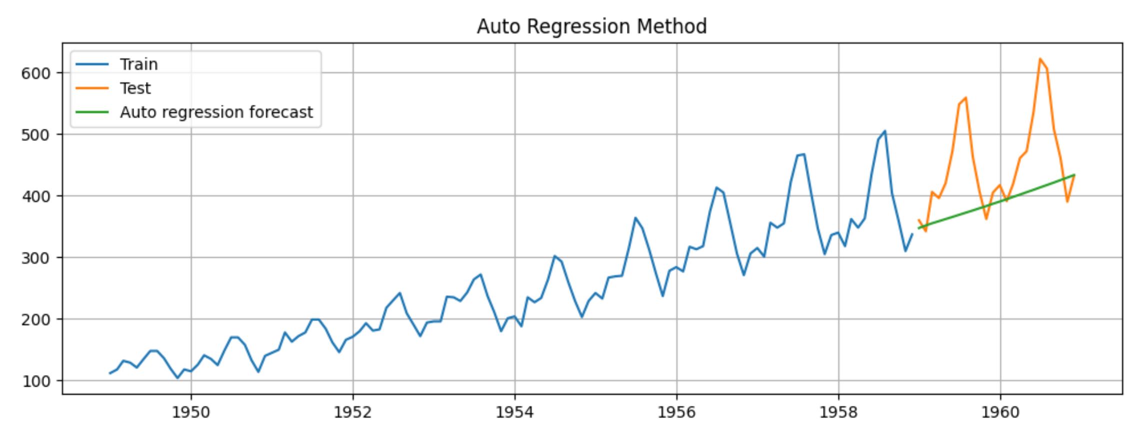
```
const     0.009473
ar.L1     0.182911
sigma2    0.010733
dtype: float64
```

```
/opt/conda/lib/python3.10/site-packages/statsmodels/tsa/base/tsa_model.py:473:
ValueWarning: No frequency information was provided, so inferred frequency MS
will be used.
  self._init_dates(dates, freq)
/opt/conda/lib/python3.10/site-packages/statsmodels/tsa/base/tsa_model.py:473:
ValueWarning: No frequency information was provided, so inferred frequency MS
will be used.
  self._init_dates(dates, freq)
/opt/conda/lib/python3.10/site-packages/statsmodels/tsa/base/tsa_model.py:473:
ValueWarning: No frequency information was provided, so inferred frequency MS
will be used.
  self._init_dates(dates, freq)
```

7.1 Recover original time series

[207]:
```
y_hat_ar = data_boxcox_diff.copy()
y_hat_ar['ar_forecast_boxcox_diff'] = model_fit.predict(data_boxcox_diff.index.
 ↪min(), data_boxcox_diff.index.max())
y_hat_ar['ar_forecast_boxcox'] = y_hat_ar['ar_forecast_boxcox_diff'].cumsum()
y_hat_ar['ar_forecast_boxcox'] = y_hat_ar['ar_forecast_boxcox'].
 ↪add(data_boxcox[0])
y_hat_ar['ar_forecast'] = np.exp(y_hat_ar['ar_forecast_boxcox'])
```

[208]:
```
plt.figure(figsize=(12,4))
plt.grid()
plt.plot(train['Passengers'], label='Train')
plt.plot(test['Passengers'], label='Test')
plt.plot(y_hat_ar['ar_forecast'][test.index.min():], label='Auto regression␣
 ↪forecast')
plt.legend(loc='best')
plt.title('Auto Regression Method')
plt.show()
```



[209]:
```
rmse = np.sqrt(mean_squared_error(test['Passengers'],␣
↪y_hat_ar['ar_forecast'][test.index.min():])).round(2)
mape = np.round(np.mean(np.abs(test['Passengers']-y_hat_ar['ar_forecast'][test.
↪index.min():])/test['Passengers'])*100,2)
```

[210]:
```
results = pd.DataFrame(columns=['Method', 'RMSE', 'MAPE'])
tempResults = pd.DataFrame({'Method':['Autoregressive (AR) method'], 'RMSE':␣
↪[rmse],'MAPE': [mape] })
results = pd.concat([results, tempResults])
results = results[['Method', 'RMSE', 'MAPE']]
results
```

[210]:

	Method	RMSE	MAPE
0	Autoregressive (AR) method	93.42	13.72

8 Moving average method (MA)

[211]:
```
model = ARIMA(train_data_boxcox_diff, order=(0, 0, 1))
model_fit = model.fit()
print(model_fit.params)
```

```
const     0.009523
ma.L1     0.258490
sigma2    0.010579
dtype: float64
```

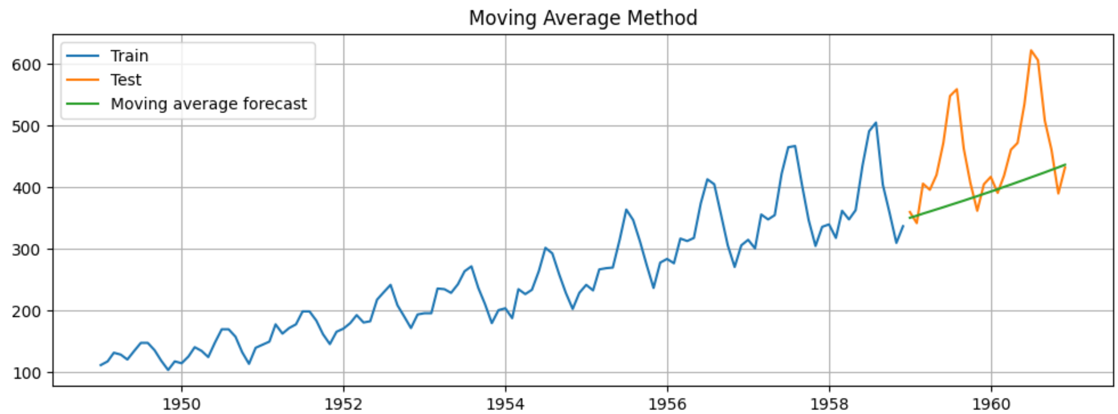
```
/opt/conda/lib/python3.10/site-packages/statsmodels/tsa/base/tsa_model.py:473:
ValueWarning: No frequency information was provided, so inferred frequency MS
will be used.
  self._init_dates(dates, freq)
/opt/conda/lib/python3.10/site-packages/statsmodels/tsa/base/tsa_model.py:473:
ValueWarning: No frequency information was provided, so inferred frequency MS
will be used.
  self._init_dates(dates, freq)
/opt/conda/lib/python3.10/site-packages/statsmodels/tsa/base/tsa_model.py:473:
ValueWarning: No frequency information was provided, so inferred frequency MS
will be used.
  self._init_dates(dates, freq)
```

8.1 Recover original time series

[212]:
```
y_hat_ma = data_boxcox_diff.copy()
y_hat_ma['ma_forecast_boxcox_diff'] = model_fit.predict(data_boxcox_diff.index.
↪min(), data_boxcox_diff.index.max())
y_hat_ma['ma_forecast_boxcox'] = y_hat_ma['ma_forecast_boxcox_diff'].cumsum()
```

```
y_hat_ma['ma_forecast_boxcox'] = y_hat_ma['ma_forecast_boxcox'].
  ↪add(data_boxcox[0])
y_hat_ma['ma_forecast'] = np.exp(y_hat_ma['ma_forecast_boxcox'])
```

[213]:
```
plt.figure(figsize=(12,4))
plt.grid()
plt.plot(data['Passengers'][:train_len], label='Train')
plt.plot(data['Passengers'][train_len:], label='Test')
plt.plot(y_hat_ma['ma_forecast'][test.index.min():], label='Moving average␣
  ↪forecast')
plt.legend(loc='best')
plt.title('Moving Average Method')
plt.show()
```



[214]:
```
rmse = np.sqrt(mean_squared_error(test['Passengers'],␣
  ↪y_hat_ma['ma_forecast'][test.index.min():])).round(2)
mape = np.round(np.mean(np.abs(test['Passengers']-y_hat_ma['ma_forecast'][test.
  ↪index.min():])/test['Passengers'])*100,2)
```

[215]:
```
tempResults = pd.DataFrame({'Method':['Moving Average (MA) method'], 'RMSE':␣
  ↪[rmse],'MAPE': [mape] })
results = pd.concat([results, tempResults])
results = results[['Method', 'RMSE', 'MAPE']]
results
```

[215]:

	Method	RMSE	MAPE
0	Autoregressive (AR) method	93.42	13.72
0	Moving Average (MA) method	91.61	13.40

9 Auto regression moving average method (ARMA)

[216]:
```
model = ARIMA(train_data_boxcox_diff, order=(1, 0, 1))
model_fit = model.fit()
print(model_fit.params)
```

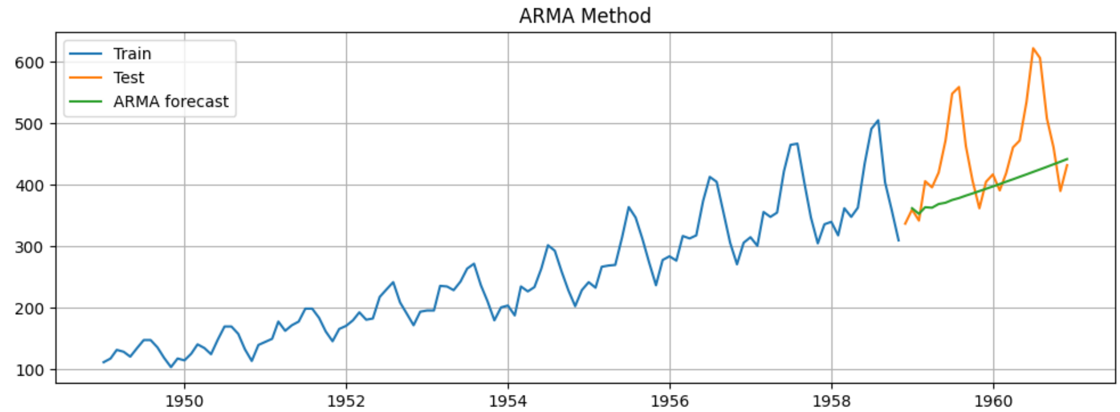
```
/opt/conda/lib/python3.10/site-packages/statsmodels/tsa/base/tsa_model.py:473:
ValueWarning: No frequency information was provided, so inferred frequency MS
will be used.
  self._init_dates(dates, freq)
/opt/conda/lib/python3.10/site-packages/statsmodels/tsa/base/tsa_model.py:473:
ValueWarning: No frequency information was provided, so inferred frequency MS
will be used.
  self._init_dates(dates, freq)
/opt/conda/lib/python3.10/site-packages/statsmodels/tsa/base/tsa_model.py:473:
ValueWarning: No frequency information was provided, so inferred frequency MS
will be used.
  self._init_dates(dates, freq)
```

```
const     0.009628
ar.L1    -0.581788
ma.L1     0.837584
sigma2    0.010129
dtype: float64
```

9.1 Recover original time series

[217]:
```
y_hat_arma = data_boxcox_diff.copy()
y_hat_arma['arma_forecast_boxcox_diff'] = model_fit.predict(data_boxcox_diff.
 ↪index.min(), data_boxcox_diff.index.max())
y_hat_arma['arma_forecast_boxcox'] = y_hat_arma['arma_forecast_boxcox_diff'].
 ↪cumsum()
y_hat_arma['arma_forecast_boxcox'] = y_hat_arma['arma_forecast_boxcox'].
 ↪add(data_boxcox[0])
y_hat_arma['arma_forecast'] = np.exp(y_hat_arma['arma_forecast_boxcox'])
```

[218]:
```
plt.figure(figsize=(12,4))
plt.grid()
plt.plot( data['Passengers'][:train_len-1], label='Train')
plt.plot(data['Passengers'][train_len-1:], label='Test')
plt.plot(y_hat_arma['arma_forecast'][test.index.min():], label='ARMA forecast')
plt.legend(loc='best')
plt.title('ARMA Method')
plt.show()
```



[219]:
```
rmse = np.sqrt(mean_squared_error(test['Passengers'],␣
  ↪y_hat_arma['arma_forecast'][train_len-1:])).round(2)
mape = np.round(np.mean(np.
  ↪abs(test['Passengers']-y_hat_arma['arma_forecast'][train_len-1:])/
  ↪test['Passengers'])*100,2)
```

[220]:
```
tempResults = pd.DataFrame({'Method':['Autoregressive moving average (ARMA)␣
  ↪method'], 'RMSE': [rmse],'MAPE': [mape] })
results = pd.concat([results, tempResults])
results = results[['Method', 'RMSE', 'MAPE']]
results
```

[220]:

	Method	RMSE	MAPE
0	Autoregressive (AR) method	93.42	13.72
0	Moving Average (MA) method	91.61	13.40
0	Autoregressive moving average (ARMA) method	88.74	12.81

10 Seasonal auto regressive integrated moving average (SARIMA)

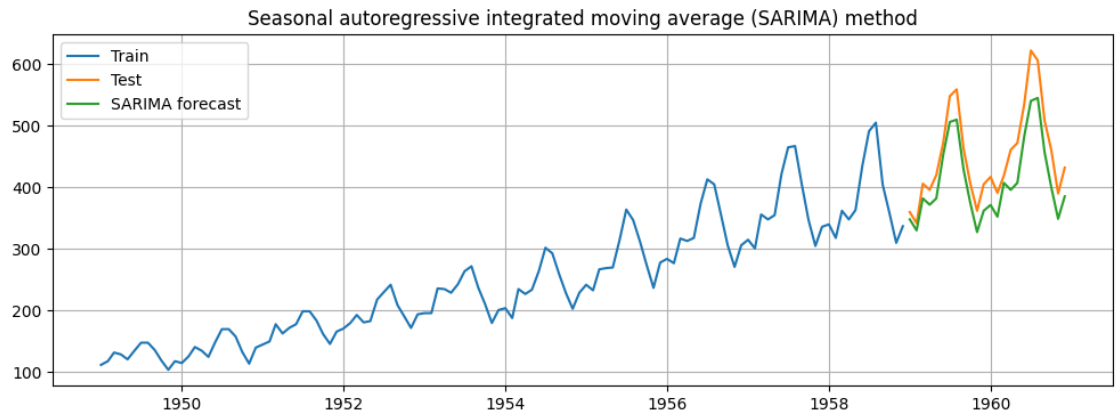
[221]:
```
from statsmodels.tsa.statespace.sarimax import SARIMAX

model = SARIMAX(train_data_boxcox, order=(1, 1, 1), seasonal_order=(1, 1, 1,␣
  ↪12))
model_fit = model.fit()
print(model_fit.params)
```

```
/opt/conda/lib/python3.10/site-packages/statsmodels/tsa/base/tsa_model.py:473:
ValueWarning: No frequency information was provided, so inferred frequency MS
will be used.
  self._init_dates(dates, freq)
/opt/conda/lib/python3.10/site-packages/statsmodels/tsa/base/tsa_model.py:473:
```

```
ValueWarning: No frequency information was provided, so inferred frequency MS
will be used.
  self._init_dates(dates, freq)
 This problem is unconstrained.

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= -1.61271D+00    |proj g|=  4.53288D+00

At iterate    5    f= -1.62802D+00    |proj g|=  1.10942D+00

At iterate   10    f= -1.63884D+00    |proj g|=  2.88919D-02

At iterate   15    f= -1.64511D+00    |proj g|=  1.67627D-01

At iterate   20    f= -1.64525D+00    |proj g|=  1.12474D-01

           * * *

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     24     37      1     0     0   5.702D-03  -1.645D+00
  F =  -1.6452969897436860

CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH
ar.L1      -0.235263
ma.L1      -0.091737
ar.S.L12   -0.071385
ma.S.L12   -0.493694
sigma2      0.001403
dtype: float64
```

[222]:
```
y_hat_sarima = data_boxcox_diff.copy()
y_hat_sarima['sarima_forecast_boxcox'] = model_fit.predict(data_boxcox_diff.
 ↪index.min(), data_boxcox_diff.index.max())
y_hat_sarima['sarima_forecast'] = np.exp(y_hat_sarima['sarima_forecast_boxcox'])
```

[223]:
```
plt.figure(figsize=(12,4))
plt.grid()
plt.plot(train['Passengers'], label='Train')
plt.plot(test['Passengers'], label='Test')
plt.plot(y_hat_sarima['sarima_forecast'][test.index.min():], label='SARIMA␣
 ↪forecast')
plt.legend(loc='best')
plt.title('Seasonal autoregressive integrated moving average (SARIMA) method')
plt.show()
```



[224]:
```
rmse = np.sqrt(mean_squared_error(test['Passengers'],␣
 ↪y_hat_sarima['sarima_forecast'][test.index.min():])).round(2)
mape = np.round(np.mean(np.
 ↪abs(test['Passengers']-y_hat_sarima['sarima_forecast'][test.index.min():])/
 ↪test['Passengers'])*100,2)
```

[225]:
```
tempResults = pd.DataFrame({'Method':['Seasonal autoregressive integrated␣
 ↪moving average (SARIMA) method'], 'RMSE': [rmse],'MAPE': [mape] })
results = pd.concat([results, tempResults])
results = results[['Method', 'RMSE', 'MAPE']]
results
```

[225]:

	Method	RMSE	MAPE
0	Autoregressive (AR) method	93.42	13.72
0	Moving Average (MA) method	91.61	13.40
0	Autoregressive moving average (ARMA) method	88.74	12.81
0	Seasonal autoregressive integrated moving aver…	44.71	8.85