Module 8: Time Series Analysis

Introduction

The problematics of a time series analysis is closely related to the regression analysis. In time series analysis, the independent variable x is always time. In the following module, we will cover the most common techniques for analysis and forecasting of time series.

Learning Outcomes

In this module, you will:

- Gain familiarity with the concept of time series
- Learn basic methods of analysis of the time series
- Learn how to forecast a time series

Reading and Resources

We invite you to further supplement this notebook with the following recommended texts:

- Hyndman, R.J. & Athanasopoulos, G. (2018). *Forecasting: Principles and Practice. OTexts. https://www.datasciencecentral.com/group/resources/forum/topics/free-book-forecasting-principles-and-practice (https://www.datasciencecentral.com/group/resources/forum/topics/free-book-forecasting-principles-and-practice)
- Bisgaard, S. & Kulahci, M. (2011). *Time series analysis and forecasting by example.* John Wiley & Sons. https://www.wiley.com/en-us/Time+Series+Analysis+and+Forecasting+by+Example-p-9780470540640 (https://www.wiley.com/en-us/Time+Series+Analysis+and+Forecasting+by+Example-p-9780470540640)

Table of Contents

Time series

A **time series** is a set of observations at different points in time. A time series can have a **fixed frequency** of observations (i.e., monthly), or have an **irregular frequency** of observations, i.e., the data points are spaced in time irregularly (whenever data is available). In this module, we will only consider time series that are observed at regular intervals of time (e.g., hourly, daily, weekly, monthly, quarterly, annually). Hereafter, we will work with **discretized time**, i.e. we will treat time as a variable, whose value grows incrementally:

$$t + 1 = t + \Delta t$$

where Δt is a time interval at which the observation of the time series, often referred to as **realizations**, are collected, and we will call it a **time step**.

Very often, we assume that the time series may involve certain periodicity (e.g. yearly period of the flu incidence rates, daily period of the temperature, etc.). In these cases, it is important to specify the time series **frequency**, i.e. number of observations within the presumed period.

For example:

- The frequency of daily observations of the data with yearly periodicity is F = 365.
- The frequency of hourly observation of the data with daily periodicity is F = 24.

NOTE:

Proper analysis of the time series requires at least two observations per period, i.e. frequency of observations ≥ 2.

EXERCISE 1

- Import the two time series datasets provided (*daily_births.csv, passengers.csv*):
  - *Births* - the total daily number of female births in California recorded across the year 1959
  - *Passengers* - the total monthly number of passengers of the international airlines (in thousands) recorded across the span of years 1949 - 1960.
- Without looking on the data, answer the following questions:
  - What is the frequency of the both time series?
  - Does either time series contain periodicity?
  - Are the time series mostly increasing/decreasing?
  - Which one of the two is most noisy?
- Plot the data to confirm/refute your answers

In [22]:
```
#your work here
```

In [ ]:
```

```

In [ ]:

In [ ]:

In [ ]:

Solution

First, we will import the libraries. Some of the libraries below will be used in the next exercises.

In [23]:
```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

from random import randrange
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.arima_model import ARIMA
from statsmodels.graphics import tsaplots
```

In [24]:
```
# Load passenger data set and save to DataFrame
passengers = pd.read_csv('./module08/passengers.csv',
                         header=0,
                         index_col=0,
                         parse_dates=True,
                         sep=';')
```

In [25]:
```
# Load daily births data set and save to DataFrame
births = pd.read_csv('./module08/daily_births.csv',
                     header=0,
                     index_col=0,
                     parse_dates=True,
                     sep=',')
```

Let's take a look on the first 5 rows of the each time series.

In [26]:
```
passengers.head()
```

Out[26]:

month	n_passengers
1949-01-01	112
1949-02-01	118
1949-03-01	132
1949-04-01	129
1949-05-01	121

In [27]:
```
births.head()
```

Out[27]:

Date	Births
1959-01-01	35
1959-01-02	32
1959-01-03	30
1959-01-04	31
1959-01-05	44

Let's check whether the time series are regular, i. e. whether the time between any two ovservations remains the same.

In [28]:
```
passengers.index.to_series().diff().value_counts()
```

Out[28]:
```
31 days    83
30 days    48
28 days     9
29 days     3
Name: month, dtype: int64
```

In [29]:
```
births.index.to_series().diff().value_counts()
```

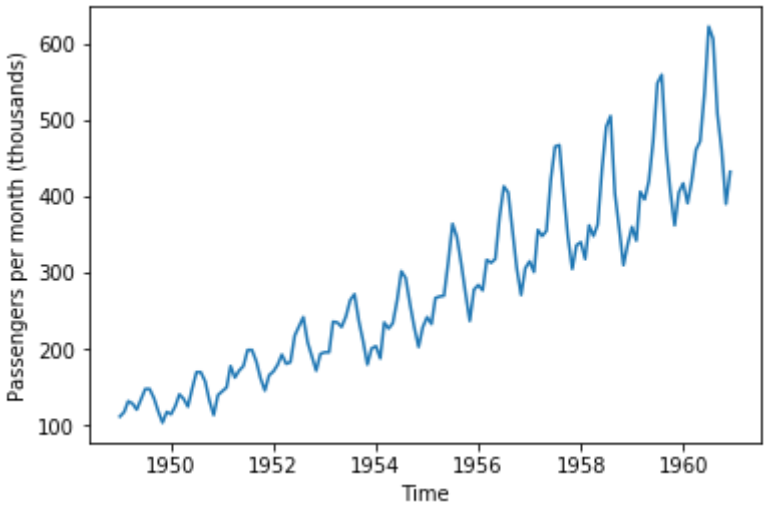
Out[29]:
```
1 days    364
Name: Date, dtype: int64
```

As we can see, the *passengers* time series contains small iregularities caused by the unequal length of the calendar months.

We will plot both time series.

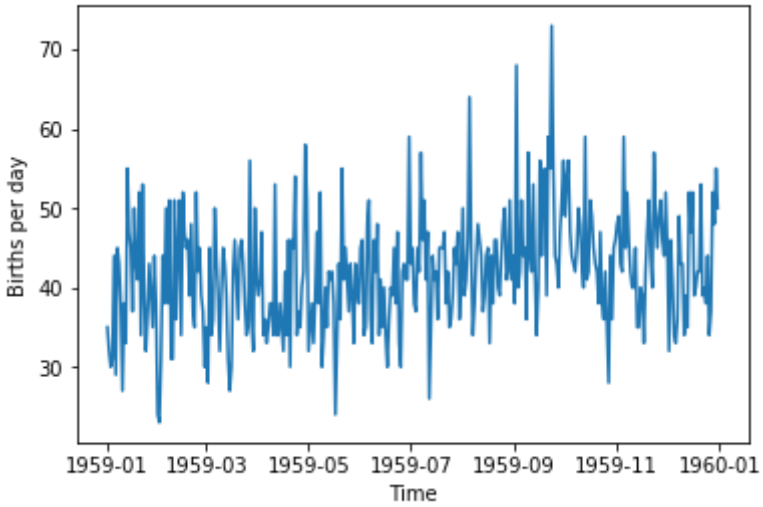
In [30]:
```
plt.plot(passengers.index,
         passengers.n_passengers)
plt.xlabel('Time')
plt.ylabel('Passengers per month (thousands)')
```

Out[30]:
```
Text(0, 0.5, 'Passengers per month (thousands)')
```



In [31]:
```
plt.plot(births.index,
         births.Births)
plt.xlabel('Time')
plt.ylabel('Births per day')
```

Out[31]:
```
Text(0, 0.5, 'Births per day')
```



Now, we can answer the above questions:

- The frequency of the *passengers* dataset is 12, corresponding to 12 observations within the period of one year.
- The frequency of the *births* datasets is 365, corresponding to 365 observations within the period of one year.
- The *passengers* series signal is clearly periodic, probably due to the seasonal changes in people's travelling habits.
- There is not obvious periodicity in the *births* time series, as expected.
- The *passengers* series signal is mostly growing, which probably reflects the global growth of air transportation during the given period of time.
- The *births* time series is clearly most noisy.

Techniques of Time Series Analysis

Time series can be subjected to various analysis, the aim of which is typically to obtain a better understanding of the underlying data generating process, or prediction of the future realization of the time series. The most common techniques used to analyse time series involve (but are not limited to):

- **Smoothing of the time series**
- **Autocorrelation analysis**
- **Time series decomposition**
- **Modelling of the time series and forecasting**

In the following section, we will describe these techniques.

Smoothing of the Time Series

Smoothing describes a variety of techniques used to reduce, or remove "noise" from the times series. The procedure typically involves calculation of some type of **moving average** (MA) and its subsequent subtraction from the original data, which effectively filters out the short-term fluctuations while preserving the long-term trends.

Simple moving average

In the simplest case, the moving average used to smooth the times series is the **simple moving average** (SMA). SMA of the signal y at time t is calculated as a mean calculated across the set of $k-1$ consecutive realizations of y preceding the time t and the realization at the time t. Mathematically, this can be formulated as:

$$\mathrm{SMA}_y(t,k) = \frac{y_t + y_{t-1} + \dots y_{t-(k-1)}}{k}$$

or, more compactly:

$$\mathrm{SMA}_y(t,k) = \frac{1}{k}\sum_{i=0}^{k-1} \quad y_{t-i}$$

where:

- y_t is the realization of the time series y at the time t.
- SMA_y is the SMA of the times series y at the time t.
- set of values $y_t + y_{t-1} + \dots y_{t-(k-1)}$ comprise the **window** of the moving average
- k is the window size, sometime referred to as **order** of the MA

Smoothed signal $y\prime$ is then produced from original time series y by applying the SMA:

$$y_t\prime = \mathrm{SMA}_y(t,k)$$

The size of the window determines "how much" of the signal is to be removed as fluctuation. The bigger the window size, the larger the fluctuations that will be filtered out and the smoother the resulting signal will be.

Exponential moving average

Sometimes (very often, in fact), to smooth the time series, it is desirable to weight the more recent realizations of the time series more than the older ones. A very popular way of doing this is to use the **exponential moving average** (EMA), which, for any given time t, so that $t > 1$, can be calculated by applying the following recursive function:

$$EMA_y(t,\alpha) = \alpha \cdot y_t + (1-\alpha)\cdot y_{t-1}$$

where:

- y_t and y_{t-1} are the realizations of the time series y at the time t and $t-1$ respectively
- $EMA_y(t)$ is the value of the EMA of y at the time t
- α is the **exponential smoothing factor**, whose values may range between 0 and 1 ($0 < \alpha < 1$), it represents the degree of weighting decrease. The greater α discounts older realizations faster.

Note, that at the time $t = 1$ the above recursive function cannot be calculated, and therefore $EMA_y(t)$ is required to be initialized by either setting y_1 , or as a mean calculated across initial realization of the y.

Similar to SMA, the smoothed signal $y\prime$ is then produced from the original time series y by applying the EMA:

$$y_t\prime = \mathrm{EMA}_y(t,\alpha)$$

NOTE:

An important consequence of the recursive character of the EMA is that, depending on the values of α and the initialization method, the early outcomes of EMA should be disregarded. These outcomes are unreliable, since EMA requires several iterations to converge to reliable values. This is sometimes called **relaxation** of the EMA.

Analogy beween EMA and SMA

There is a loose analogy between the k of SMA and α of the EMA, as similar to k, bigger values of α will yield smoother resulting signal. The differences between the orignal and smoothed signal (sometimes referred to as errors) of EMA(t, α) and SMA(t, k) have roughly the same distribution when $\alpha = 2/(k + 1)$.

Importantly, EMA takes into account all past data, whereas SMA only takes into account k past data points. Another important difference between EMA and SMA is that calculation of SMA requires the past k signal realizations, whereas calculation of EMA only requires the most recent realization and preceeding value of EMA, which makes EMA less memory demanding and computationally "cheaper".

There are numerous modifications of SMA and EMA as well as their alternatives, for more details about these we recommend you to visit: https://en.wikipedia.org/wiki/Moving_average (https://en.wikipedia.org/wiki/Moving_average) (Moving Average, n.d.).

EXERCISE 2:

- Apply simple and exponential moving average to smooth the *births* time series, using different parameters.
- Does the number of births across the weekdays difer from the average?

In [32]:
```
#your work here
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

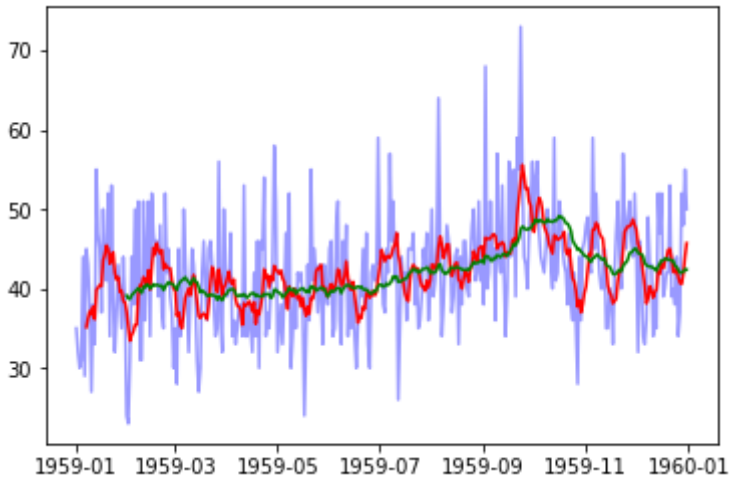
Solution

We will first calculate the simple moving average. We will use the pandas *rolling* function, using two different windows sizes: 7 days and 30 days.

In [35]:
```
# Calculate the moving average
rol_short = births.Births.rolling(window = 7)
rol_long = births.Births.rolling(window = 31)

# Add both to the data frame as a new columns
births['SMA_short'] = rol_short.mean()
births['SMA_long'] = rol_long.mean()
```

In [36]:
```
# Plot simple moving averages along the original signal
plt.plot(births.index,
         births.Births,
         color = 'blue',
         alpha = 0.4)

plt.plot(births.index,
         births.SMA_short,
         color = 'red')

plt.plot(births.index,
         births.SMA_long,
         color = 'green')
```

Out[36]:
```
[<matplotlib.lines.Line2D at 0x1c21ef6c18>]
```

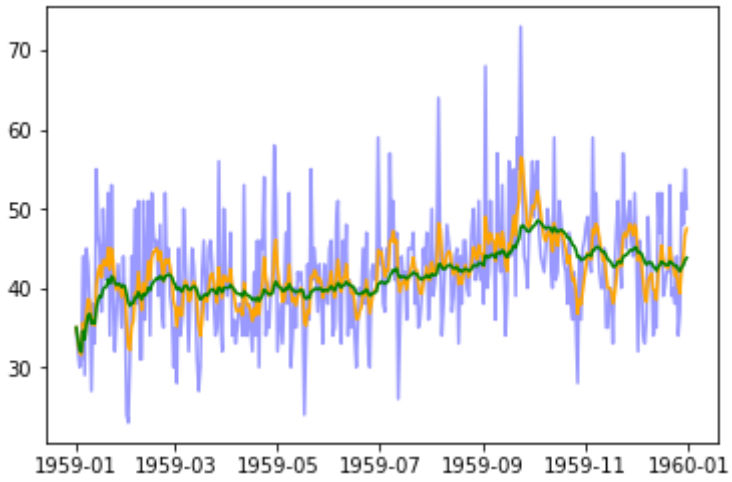


We will calculate exponential moving average using the pandas *ewm* function. We will use long and short memory smoothing with parameter α equal 0.25 and 0.05 respectively.

In [64]:
```
births['EMA_short'] = births.Births.ewm(alpha=0.25, adjust=True).mean()
births['EMA_long'] = births.Births.ewm(alpha=0.05).mean()
```

In [65]:
```
plt.plot(births.index,
         births.Births,
         color = 'blue',
         alpha = 0.4)

plt.plot(births.index,
         births.EMA_short,
         color = 'orange')

plt.plot(births.index,
         births.EMA_long,
         color = 'green')
```

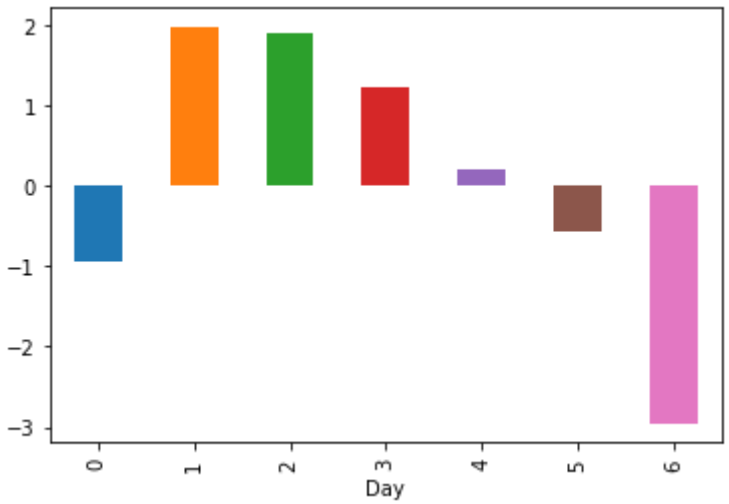
Out[65]:
```
[<matplotlib.lines.Line2D at 0x1c24080cf8>]
```



Let's take a look if the number of births across the weekdays differ from the short moving average.

In [66]:
```
# Calculate the difference from the short 7 day moving average
births['Diff'] = births.Births - births.SMA_short

# Identify day of the week (0 - Monday, 6 - Sunday)
births['Day'] = births.index.dayofweek

# Plot the results
births.Diff.groupby(births.Day).mean().plot.bar()
```

Out[66]: <matplotlib.axes._subplots.AxesSubplot at 0x1c24103710>



In [67]:
```
births.shape
```

Out[67]: (365, 7)

In [68]:
```
births
```

Out[68]:

Date	Births	SMA_short	SMA_long	EMA_short	EMA_long	Diff	Day
1959-01-01	35	NaN	NaN	35.000000	35.000000	NaN	3
1959-01-02	32	NaN	NaN	33.285714	33.461538	NaN	4
1959-01-03	30	NaN	NaN	31.864865	32.248028	NaN	5
1959-01-04	31	NaN	NaN	31.548571	31.911621	NaN	6
1959-01-05	44	NaN	NaN	35.629962	34.583451	NaN	0
1959-01-06	29	NaN	NaN	33.613603	33.529605	NaN	1
1959-01-07	45	35.142857	NaN	36.898711	35.430800	9.857143	2
1959-01-08	43	36.285714	NaN	38.593726	36.555230	6.714286	3
1959-01-09	38	37.142857	NaN	38.433245	36.750600	0.857143	4
1959-01-10	27	36.714286	NaN	35.404367	35.535612	-9.714286	5
1959-01-11	38	37.714286	NaN	36.081891	35.821371	0.285714	6
1959-01-12	33	36.142857	NaN	35.286214	35.514460	-3.142857	0
1959-01-13	55	39.857143	NaN	40.334596	37.516435	15.142857	1
1959-01-14	47	40.142857	NaN	42.031177	38.441977	6.857143	2
1959-01-15	45	40.428571	NaN	42.783435	39.052925	4.571429	3
1959-01-16	37	40.285714	NaN	41.322938	38.869587	-3.285714	4
1959-01-17	50	43.571429	NaN	43.508634	39.826006	6.428571	5
1959-01-18	43	44.285714	NaN	43.380754	40.089283	-1.285714	6
1959-01-19	41	45.428571	NaN	42.783038	40.162416	-4.428571	0
1959-01-20	52	45.000000	NaN	45.094609	41.085044	7.000000	1
1959-01-21	34	43.142857	NaN	42.314344	40.547841	-9.142857	2
1959-01-22	53	44.285714	NaN	44.990532	41.468224	8.714286	3
1959-01-23	39	44.571429	NaN	43.490893	41.290050	-5.571429	4
1959-01-24	32	42.000000	NaN	40.615284	40.633983	-10.000000	5
1959-01-25	37	41.142857	NaN	39.710782	40.382534	-4.142857	6
1959-01-26	43	41.428571	NaN	40.533551	40.560236	1.571429	0
1959-01-27	39	39.571429	NaN	40.150001	40.456172	-0.571429	1
1959-01-28	35	39.714286	NaN	38.862092	40.098237	-4.714286	2
1959-01-29	44	38.428571	NaN	40.146875	40.350268	5.571429	3
1959-01-30	38	38.285714	NaN	39.610060	40.200638	-0.285714	4
...	...	...	...	...	...	...	...
1959-12-02	32	45.571429	44.483871	43.439594	44.456671	-13.571429	2
1959-12-03	46	45.142857	44.387097	44.079695	44.533838	0.857143	3

Date	Births	SMA_short	SMA_long	EMA_short	EMA_long	Diff	Day
1959-12-04	41	43.714286	44.322581	43.309771	44.357146	-2.714286	4
1959-12-05	34	42.000000	44.064516	40.982329	43.839289	-8.000000	5
1959-12-06	33	40.428571	43.225806	38.986746	43.297324	-7.428571	6
1959-12-07	36	38.142857	42.935484	38.240060	42.932458	-2.142857	0
1959-12-08	49	38.714286	42.838710	40.930045	43.235835	10.285714	1
1959-12-09	43	40.285714	42.741935	41.447534	43.224043	2.714286	2
1959-12-10	43	39.857143	42.774194	41.835650	43.212841	3.142857	3
1959-12-11	34	38.857143	42.580645	39.876738	42.752199	-4.857143	4
1959-12-12	39	39.571429	42.548387	39.657553	42.564589	-0.571429	5
1959-12-13	35	39.857143	42.225806	38.493165	42.186360	-4.857143	6
1959-12-14	52	42.142857	42.774194	41.869874	42.677042	9.857143	0
1959-12-15	47	41.857143	43.161290	43.152405	42.893190	5.142857	1
1959-12-16	52	43.142857	43.548387	45.364304	43.348530	8.857143	2
1959-12-17	39	42.571429	43.548387	43.773228	43.131104	-3.571429	3
1959-12-18	40	43.428571	43.774194	42.829921	42.974548	-3.428571	4
1959-12-19	42	43.857143	43.774194	42.622441	42.925821	-1.857143	5
1959-12-20	42	44.857143	43.612903	42.466831	42.879530	-2.857143	6
1959-12-21	53	45.000000	43.677419	45.100123	43.385554	8.000000	0
1959-12-22	39	43.857143	43.516129	43.575092	43.166276	-4.857143	1
1959-12-23	40	42.142857	43.516129	42.681319	43.007962	-2.142857	2
1959-12-24	38	42.000000	42.903226	41.510989	42.757564	-4.000000	3
1959-12-25	44	42.571429	42.741935	42.133242	42.819686	1.428571	4
1959-12-26	34	41.428571	42.387097	40.099932	42.378701	-7.428571	5
1959-12-27	37	40.714286	42.000000	39.324949	42.109766	-3.714286	6
1959-12-28	52	40.571429	42.032258	42.493711	42.604278	11.428571	0
1959-12-29	48	41.857143	42.096774	43.870284	42.874064	6.142857	1
1959-12-30	55	44.000000	42.451613	46.652713	43.480361	11.000000	2
1959-12-31	50	45.714286	42.387097	47.489535	43.806343	4.285714	3

365 rows × 7 columns

As we can see, the number of births is lower on weekends and Mondays.

Autocorrelation Analysis

Autocorrelation, sometimes referred to as serial correlation, *is a correlation between the actual values of the time series and its preceding values*, as a function of the delay between them. Mathematically, this is expressed as:

$$R_{yy}(\tau) = \frac{E\Big[(y_t - \mu)(y_{t+\tau} - \mu)\Big]}{\sigma^2}$$

where:

- y is the time series, sometimes called **signal**
- R_{yy} denotes autocorrelation of the time series y
- τ is a delay between the actual values and preceding ones; and is often called **lag**
- y_t, $y_{t+\tau}$ are realizations, of the time series y recorded at the time t and $t + \tau$, respectively
- E is the **expected value** operator. It is *de facto* long-time average of the argument: $(y_t - \mu)(y_{t-\tau} - \mu)$
- μ and σ are the mean and standard deviation of the y.

Removing the normalization term σ^2, we would obtain **autocovariance**, which is frequently used as an alternative to autocorrelation:

$$C(\tau) = E[(y_t - \mu)(y_{t-\tau} - \mu)]$$

NOTE:

In some literature, the term autocorrelation and autocovariance are used interchangeably. However, there are at least two substantial differences between them:

- Autocovariance is resistant to cases when $\sigma = 0$, which makes it useful in cases when we don't know σ prior the calculation of the autocorrelation, i.e. when there is a chance that the time series is constant.
- Autocorrelation, in contrast to autocovariance is normalized, i.e. can be used for comparison of time series with very different range of values (e.g. record of two different stocks)

We need to mention that autocorrelation is a special case of more general function called **cross-correlation** $R_{yy}(\tau)$:

$$R_{xy}(\tau) = \frac{E\Big[(x_t - \mu_x)(y_{t+\tau} - \mu_y)\Big]}{\sigma_x \sigma_y}$$

Properties of autocorrelation

There are several interesting properties of autocorrelation that are worth mentioning:

- $R_{yy}(\tau)$ is an even function, which means that $R_{yy}(\tau) = R_{yy}(-\tau)$.
- The autocorrelation of continuous signal reaches its peak at $\tau == 0$, and for any value of τ: $|R_{yy}(\tau)| < R_{yy}(0)$.
- The autocorrelation of a periodic signal is always periodic with the same period.
- The autocorrelation of the sum of two completely uncorrelated signals (the cross-correlation is equal to zero for all values of τ) equals the sum of the autocorrelations of each signal separately.
- If the autocorrelation of a signal is equal to zero for all $\tau > 0$, we say the signal is a **white noise**.

How to estimate autocorrelation

In cases, when the time is discretized and the values of μ and σ can be calculated beforehand, the autocorrelation can be estimated by the following formula:

$$\hat{R}_{yy}(k) = \frac{1}{(n-k)\sigma^2} \sum_{t=1}^{n-k} (y_t - \mu)(y_{t+k} - \mu)$$

where:

- k is the lag, i. e. number of times intervals between the actual and preceding realizations of y (discrete analogy to τ).
- n is the number of realizations in the given sample, i.e. number of data points over which we are going to caculate the autorcorrelation
- μ and σ are mean and standard deviation of the y calculated across the sample

Partial autocorrelation

Partial autocorrelation (Φ) measures the autocorrelation of one variable at the given lag, after removing the effect of all shorter lags.

For example, the partial autocorrelation of the lag $\tau = 2$ measures correlation between y_t and y_{t-2} after removing the effect of y_{t-1} on both y_t an y_{t-1}. There multiple ways how partial correlation can be defined. Probably the simplest formulation is the following recursive scheme:

$$\Phi_t(\tau) = \frac{R_t(\tau) - \sum_{j=1}^{\tau-1} \Phi_{t-1}(j) \cdot R_t(\tau - 1)}{1 - \sum_{j=1}^{\tau-1} \Phi_{t-1}(j) \cdot R_t(\tau - 1)}$$

where:

- $R_t(\tau)$ is the autocorrelation, as measured at the time t with lag τ.
- $\Phi_t(\tau)$ is the partial autocorrelation, as measured at time t with lag τ.

Correlogram

Plotting the dependence of $R_{yy}(\tau)$ on τ, we will obtain what is called **correlogram**, or **lag plot**. A correlogram is an extremely useful tool for visual examination of the properties of the time series. Correlograms can be used to assess various important properties of time series, including:

- Is the observed signal **random**?
- Is the observed signal **periodical**?
- Is the observed signal **autoregressive** - can be modeled by autoregression (see following sections)
- Etc.

EXERCISE 3:

- Use the *passengers* and *births* time series to calculate and plot:
  - Autocorrelation
  - Partial autocorrelation
- Answer the following questions:
  - International flights experienced extremely high numbers of passengers as compared to expectations for the given month. What does it imply for the number of passengers in the next consecutive 12 months?
  - Is there an association between the number of births in a given day and the week prior? If yes, what would be the explanation (consider the results of the previous exercises)?

```
In [17]: #your work here
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

Solution

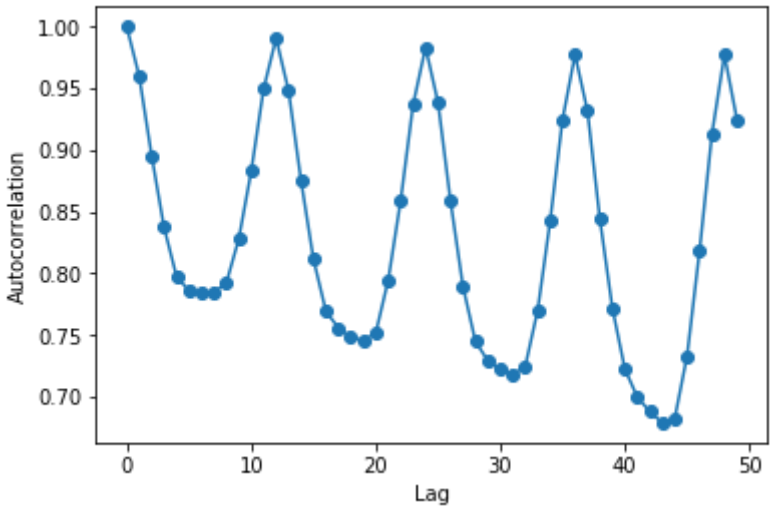
We will start with calculating the simple autocorrelation of both series

In [18]:
```
# Set the number of lags
max_lag = 50
```

In [19]:
```
# Calculate passengers autocorrelation
# using the lags ranging from 1 to 50 months
passengers_ac = [passengers.n_passengers.autocorr(lag=i) for i in range(
max_lag)]

# Calculate births autocorrelation
# using the lags ranging from 1 to 50 days
births_ac = [births.Births.autocorr(lag=i) for i in range(max_lag)]
```

In [20]:
```
# Plot the results
plt.plot(range(max_lag), passengers_ac, 'o-')
plt.xlabel('Lag')
plt.ylabel('Autocorrelation')
```

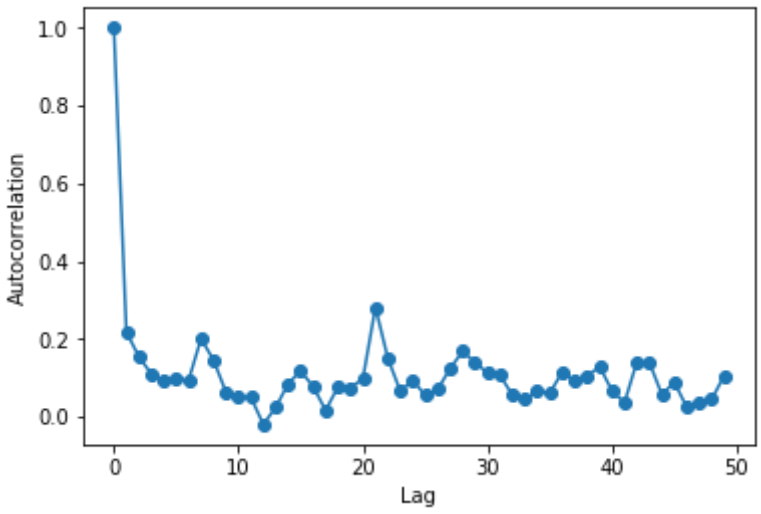
Out[20]:
```
Text(0, 0.5, 'Autocorrelation')
```



As can be seen, the signal in passengers time series is strongly correlated with its previous values from the 12, 24, 36 and 48 moths ago. This indicates the annual periodicity of the signal.

In [21]:
```
# Plot the results
plt.plot(range(max_lag), births_ac, 'o-')
plt.xlabel('Lag')
plt.ylabel('Autocorrelation')
```

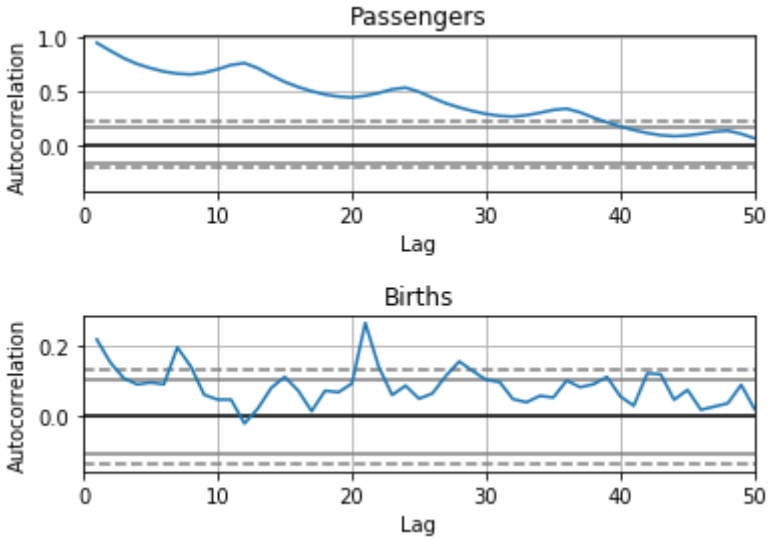
Out[21]:
```
Text(0, 0.5, 'Autocorrelation')
```



The *birth* time series shows mild correlation of with the value preceding the recent one by 21 days.

A more straightforward way of plotting the autocorrelation is using the built-in pandas function:

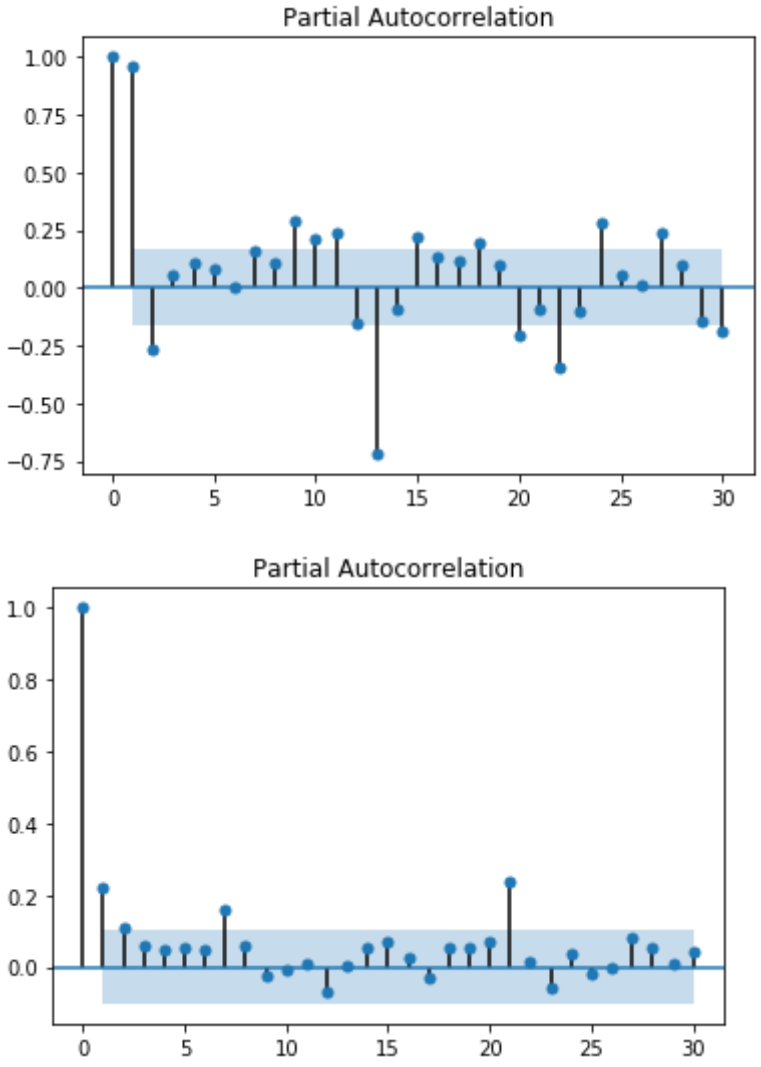
In [22]:
```
fig, (ax1, ax2) = plt.subplots(2, 1)
pd.plotting.autocorrelation_plot(passengers.n_passengers, ax = ax1)
pd.plotting.autocorrelation_plot(births.Births, ax = ax2)

ax1.set_title('Passengers')
ax2.set_title('Births')

ax1.set_xlim(0, max_lag)
ax2.set_xlim(0, max_lag)

fig.subplots_adjust(hspace = 0.8)
plt.show()
```



Note, the grey horizontal lines displayed in the plot correspond to 95% and 99% confidence bands. The solid line is 95% and dashed line is 99% confidence band. In the *passengers* time series, we observe signficant autocorrelation up to the lag corresponding to the 38 months. In the *births* dataset, the statistically significant values are only associated with lags equal to 7, 21 and 28 days. Indicating weekly periodicity if the signal.

We will calculate the partial correlations. This can be done using the functions provided by the *tsaplots* package of the *statsmodels* library.

In [23]:
```
tsaplots.plot_pacf(passengers.n_passengers, lags=30)
tsaplots.plot_pacf(births.Births, lags=30)
plt.show()
```



The blue shaded area indicates the confidence intervals, suggesting that correlation values outside of this area are very likely a correlation and not a statistical fluke. By default, this is set to a 95% confidence interval.

As we can see, partial autocorrelaton tells a slightly different story as compared to the previously used autocorelation. We can answer the above questions:

- The *passengers* signal values are positively correlated with the preceding previous ones (lag equal to 1), and negatively correlated with the values 2 and 13 months prior. Therefore an extremely high number of passengers in the given month may imply decline in the following month and possible (with smaller effect) in 13 months from now.
- The *births* signal value show significant correlation with the values 7, 14 and 21 days prior. This may be explained by clinicians scheduling the births requiring special care (e.g. c-sections, induced births, etc.) primarily to the certain days of the week, and across the month (see previous results).

Time Series Decomposition

The time series decomposition is a technique that deconstructs the observed time series into several time series components, each of which has certain characteristic properties. Time series are usually decomposed into three components:

- **Trend** ($T(t)$) - reflects the long-term progression of the time series. A trend component is present if there is a persistent increase or decrease of the signal.
- **Seasonality** or seasonal component ($S(t)$) - reflects the repeating changes of the time series. A seasonal component is present if the signal changes in a repeating cycles with a fixed period. This is when time series is affected by some periodic factors (e.g. seasonality thorough the year, day and night periodicity, etc.).
- **Noise**, or irregular component ($I(t)$) - reflects the irregularities of the time series. Noise represents the residuals of the observed time series after the other two components have been extracted.

In some cases decomposition of the time series involves also the fourth component, often referred to as **cyclicality**, or **cyclic** component ($C(t)$), that reflects the repeating changes of the times series that, in contrast to those reflected by seaonality, don't have a fixed period. Here we will limit ourselves to cases when the time series is decomposed only into three above mentioned components, ignoring the cyclic component.

A time series is considered to be a combination of these three components. This combination can be either **additive** or **multiplicative**. Additive decomposition (also known as **additive model**) assumes the observed time series to be a sum of the individual components:

$$y_t = T(t) + S(t) + I(t)$$

Multiplicative decomposition (also known as **multiplicative model**) assumes the observed time series to be a product of the individual components:

$$y_t = T(t) \times S(t) \times I(t)$$

The additive decomposition is conducted as follows:

1. If frequency of the time series, F, is an even number, calculate the estimate of the trend component $\hat{T}(t)$, by calculating the $2 \times F$ moving average. If F is an odd number, estimate the trend by calculating the F moving average.
2. Calculate the **detrended** time series by subtracting the trend estimate: $y_t - \hat{T}(t)$
3. Average the detrended values across the presumed period, i.e. calculate the average across the sets of F consecutive values of $y_t - \hat{T}(t)$ to obtain the estimate of the seasonality $\hat{S}(t)$.
4. Calculate the noise components by subtracting the trend and seasonality estimates: $\hat{I}(t) = y_t - \hat{T}(t) - \hat{S}(t)$.

NOTE:

The multiplicative decomposition is conducted by an analogous procedure, but division is applied instead of subtraction.

When to use additive and multiplicative decomposition

In general, the additive model is more appropriate if the magnitude of the seasonal component does not vary much with the actual signal. When there is an association between the two, the multiplicative model is more appropriate. Additive decomposition is more common when working with agricultural, epidemiological, or other biological time series. Multiplicative decompositions is more common with financial and economic time series.

Seasonality adjusted time series

Sometimes, the variation of the time series due to seasonality is not of interest. For example:

- Monthly unemployment rates are usually affected by seasonal variation (e.g. an increase in unemployment due to school leavers seeking work), thus complicating the ability to assess the underlying state of the labour market.
- Flu rates are heavily affected by the seasonality (e.g. high incidence rate during the cold months), which complicates assessment of the underlying epidemiologic situation.

Therefore time series are often adjusted by removing the seasonal component from the original time series, giving the so called **seasonally adjusted signal**. For an additive decomposition, the seasonally adjusted signal is given by $y_t - S(t)$, and for a multiplicative model, the seasonally adjusted signal is obtained by $y_t / S(t)$.

Measuring the strength of trend and seasonality

Sometimes we want to know how "strong" the effect of trend and/or seasonality is in our time series. The strength of a trend can be quantified by assessing its contribution to the overall variance of the signal. For a strongly trended signal, the seasonally adjusted data should have more variation than the remainder component. For the additive model, this can be mathematically formulated as follows:

$$A_T = \max\left(0, 1 - \frac{\mathrm{Var}\left(I(t)\right)}{\mathrm{Var}\left(T(t) + I(t)\right)}\right)$$

Analogously, strength of seasonality can be formulated as:

$$A_S = \max\left(0, 1 - \frac{\mathrm{Var}\left(I(t)\right)}{\mathrm{Var}\left(S(t) + I(t)\right)}\right)$$

where:

- A_T and A_S are a strength of trend and seasonality, respectively
- $T(t)$ is a trend
- $S(t)$ is a seasonality
- $I(t)$ is a noise (irregularity component)

For the multiplicative models, denominators in the above equations contain multiplication instead of addition.

EXERCISE 4:

Using the *passengers* and *births* datasets:

- Decompose the signal into trend, seasonality and noise and plot the results
- Decide which of the two models (additive or multiplicative) provide better decomposition of the signal
- Remove the trend from the time series to see how the signal changes. Repeat by removing the seasonality.
- Assess how strong the contribution of trend and seasonality is to the signal

In [24]:
```
#your work here
```

In [ ]:

In [ ]:

In [ ]:

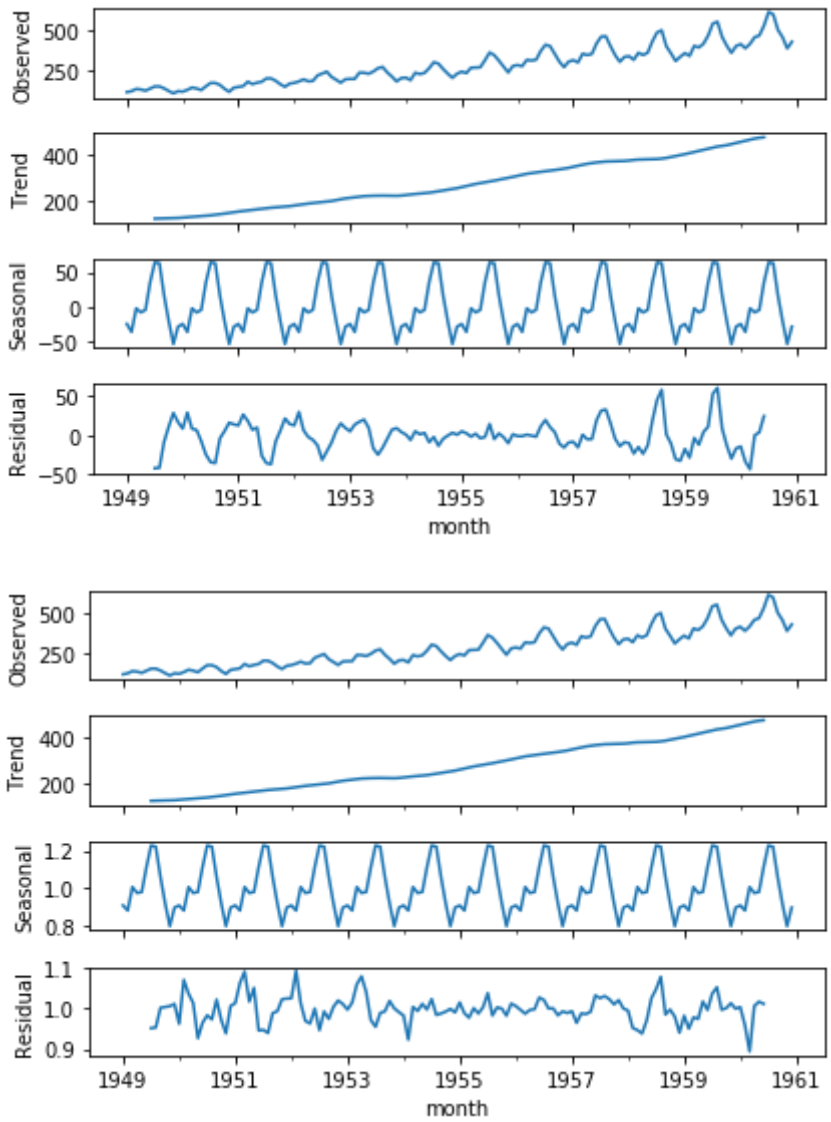
Solution

We will first decompose the signal of the *passengers* time series. To do this we will use the *statsmodels* function *seasonal_decompose*.

In [25]:
```
# Additive model
passengers_add = seasonal_decompose(passengers.n_passengers,
                                    model='additive')

# Multiplicative model
passengers_mlt = seasonal_decompose(passengers.n_passengers,
                                    model='multiplicative')
```

We will plot the decomposed signal

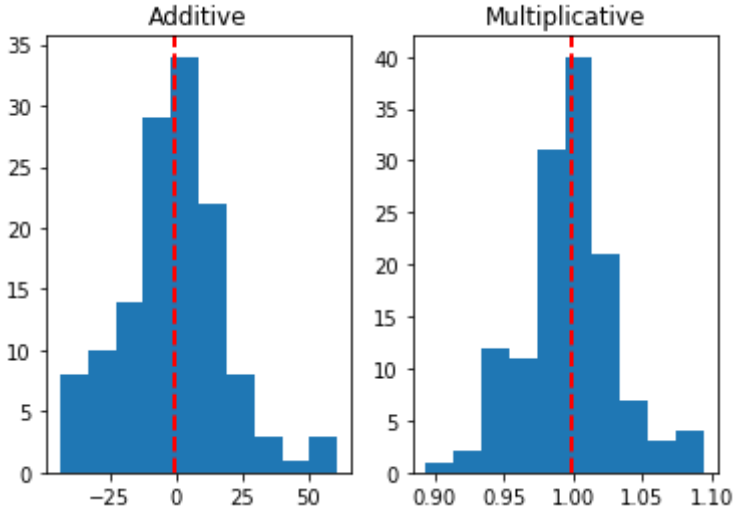
In [26]:
```
p1 = passengers_add.plot()
p2 = passengers_mlt.plot()
plt.show()
```



On first look, additive and multiplicative models produce similar results.

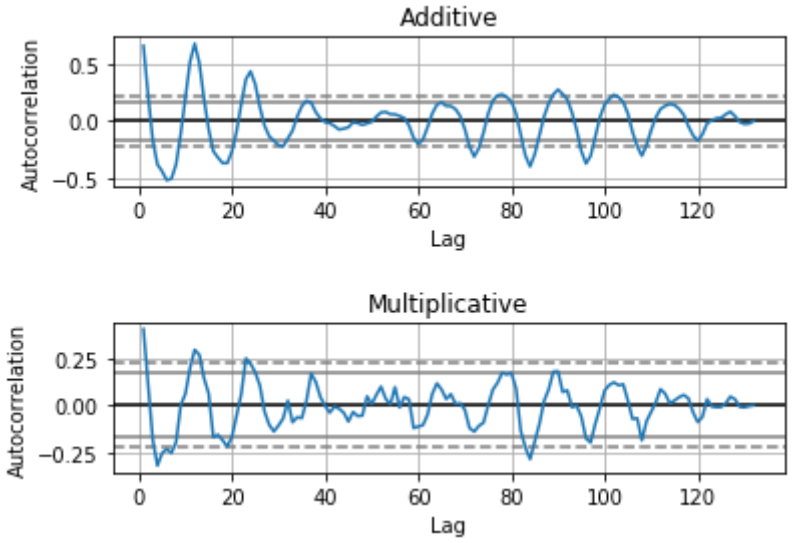
One way to decide which model is better in this case, is to examine the properties of the white noise component (here called residuals). Ideally, the noise should be white noise, i.e. independent normally distributed signal with zero mean.

In [27]:
```
# Take the residuals of both models and remove NaNs
r1 = passengers_add.resid.dropna()
r2 = passengers_mlt.resid.dropna()

# Calculate means
mu1 = r1.mean()
mu2 = r2.mean()

# Plot histograms
fig, (ax1, ax2) = plt.subplots(1, 2)
ax1.hist(r1)
ax2.hist(r2)

# Add titles
ax1.set_title('Additive')
ax2.set_title('Multiplicative')

# Add vertical lines to highlight the means
ax1.axvline(x = mu1,
            color='r',
            linestyle='dashed',
            linewidth=2)

ax2.axvline(x = mu2,
            color='r',
            linestyle='dashed',
            linewidth=2)

plt.show()
```



As can be seen, the residuals of the additive model have a much "broader" distribution. Now we can assess the independence of the residuals by calculating their autocorrelation.

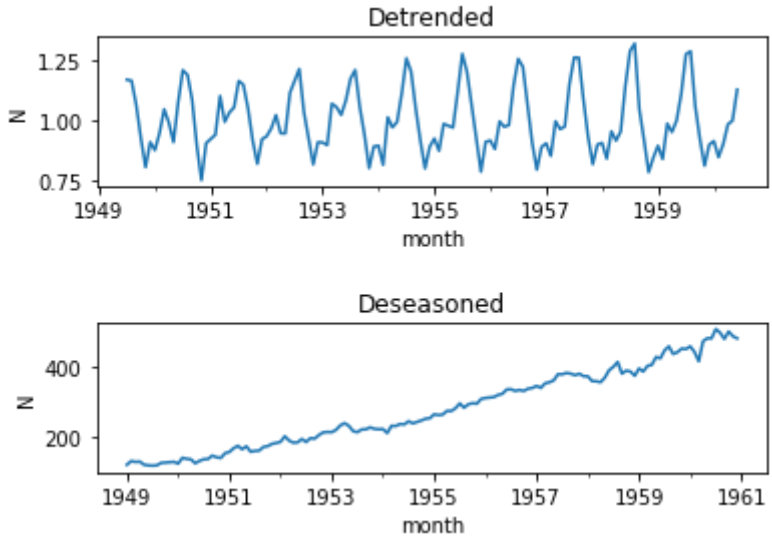
In [28]:
```
fig, (ax1, ax2) = plt.subplots(2, 1)

pd.plotting.autocorrelation_plot(r1, ax = ax1)
pd.plotting.autocorrelation_plot(r2, ax = ax2)

# Add titles
ax1.set_title('Additive')
ax2.set_title('Multiplicative')

plt.subplots_adjust(hspace = 0.9)
plt.show()
```



In both cases we observe that the noise shows some correlation with its previous values. However, in the case of the additive model, the autocorrelation is much stronger. We can conclude that the multiplicative model is the better alternative.

Next, we will remove the trend from the signal. Using the multiplicative model, we divide the signal by the trend component.

In [29]:
```
passengers['detrended'] = passengers.n_passengers / passengers_mlt.trend
passengers['deseasoned'] = passengers.n_passengers / passengers_mlt.seas
onal

fix, (ax1, ax2) = plt.subplots(2, 1)
passengers['detrended'].plot(ax = ax1)
passengers['deseasoned'].plot(ax = ax2)

ax1.set_ylabel('N')
ax2.set_ylabel('N')

ax1.set_title('Detrended')
ax2.set_title('Deseasoned')

plt.subplots_adjust(hspace = 0.9)
plt.show()
```



In [30]:
```
trend_strength = 1. - np.var(passengers_mlt.resid) / np.var(passengers_m
lt.resid * passengers_mlt.trend)
seasonality_strength = 1. - np.var(passengers_mlt.resid) / np.var(passen
gers_mlt.resid + passengers_mlt.seasonal)
```

In [31]:
```
print ('Strength of trend is: %1.3f' % trend_strength)
print ('Strength of seasonality is: %1.3f' % seasonality_strength)
```

```
Strength of trend is: 1.000
Strength of seasonality is: 0.936
```

See that the strength of the trend is greater than the strength of seasonality.

Analogous analysis can be conducted for the *births* time series. Please try this on your own.

Modelling of the Time Series and Forecasting

Naturally, people are often very interested in how the time series will continue to evolve in the future. There are numerous methods that have been developed to model the behavior of the times series in order to predict their future. Most of these methods utilize either regression analysis, some type of moving average, or a combination of both. In the following section, we will discuss the most fundamental of these methods.

Auto-regression models

Time series regression is the application of the regresssion analysis on time series data. The most common type of time series regression is the linear regression, where the response is the actual value of the time series, predictors (features) are its preceding values. This type regression is called **autoregression**, as it uses the historical values of the time series to estimate the value of the same time-series.

The autoregression can be mathematically expressed as follows:

$$\hat{y}_t = \beta_0 + \beta_1 y_{t-1} + \beta_2 y_{t-2} + \ldots + \beta_p y_{t-p} + e_t$$

or, more compactly:

$$\hat{y}_t = \beta_0 + \sum_{i=1}^{p} \beta_i y_{t-i} + e_t$$

where:

- $\hat{y}$ is the **forecasted value** (or simply forcast) of the signal y for the time t
- β_0 is the **autoregression constant**. It is equivalent of the intercept term in previously described regression analysis.
- β_i for $i = 1, 2, \ldots, p$ are the **autoregression coefficients**
- p is the **order of the autoregression**, indicating the **maximum lag** of the autoregression.
- e_t is called **white noise**. It is the **error term** equivalent to those of the regression analysis. *White noise can be thought of as the time series, whose value are normally distributed with mean equal to zero.*

Estimates of the autoregression coefficients constitute **autoregression model (AR)** of the p-th order, often denotes as $AR(p)$. The fact that the autoregression model is used to predict the future values is then expressed as:

$$\hat{y}_t = AR(p)$$

The choice of the maximum lag

An important aspect of autoregression is **the choice of the maximum lag** p (order of the autoregression), i.e. how far in to past the autoregression goes to predict the future. The maximum lag can be estimated using the **partial autocorrelation** of the times series.

Typically, as the lag increases, the absolute value of the partial autocorrelation quickly decreases, approaching zero. *The lag beyond which all the partial autocorrelations are equal (or very close) to zero can be used as the maximum lag of the autoregression/moving average.* However, generally it is safer to use lag that is incremented by one time step from this. Mathematically:

$$p = k + 1,\ |\Phi(i)| = 0\ ; \forall i > k$$

Moving average model

The **moving average (MA) model** is another common approach for time series forecasting which, rather than using the past values of the forecast time series, uses it's past forecast errors in a regression-like model:

$$\hat{y}_t = \mu_y + \epsilon_t + \theta_1 \epsilon_{t-1} + \ldots + \theta_q \epsilon_{t-q}$$

or, more compactly:

$$\hat{y}_t = \mu_y + e_t + \sum_{i=1}^{q} \theta_i \epsilon_{t-i}$$

where:

- $\hat{y}$ is the **forecasted value** of the signal y for the time t
- μ_y is the mean of the time series y.
- θ_i for $i = 1, 2, \ldots, q$ are the **model parameters**
- q is the **order of the moving average**.
- ϵ_{t-i} for $i = 1, 2, \ldots, q$ are the white noise **error terms** obtained from the previous forecasts, sometimes called **random shocks**. They are typically assumed to come from the normal distribution with zero mean.
- e_t is the error term (white noise) of the actual prediction, equivalent of the e_t of autoregression.

NOTE:

Error terms of the moving average model are different from those of autoregression model. Unlike error terms of the autoregression, they are affected by the preceding predictions. Hence different notation.

The fact that the moving average model of the q-th order is used to predict the future values is expressed as:

$$\hat{y}_t = MA(q)$$

A few important notes about the moving average model

- **IMPORTANT:** The MA model assumes the time series is **stationary**, i.e. its mean and standard deviation don't change very much over time. This is a very strong assumption that typically holds only temporarily. Therefore, MA models have only a limited "life span" and after some time, need to be re-estimated.
- Conceptually, the moving average model is a linear regression, but, because we do not observe the ϵ values directly, *it is not really a linear regression in the usual sense*.
- Error terms from the previous time step **propagate** to future predictions, as they become predictors in the next time step. Hence, each error term affects only the next q consecutive estimates of y. *This gives MA models substantial advantage of the AR models, as even the major irregularities (spikes, dropdowns) of the time series affect the predictions only in next few steps.*
- The name "moving average model" comes from the fact that each value of $\hat{y}$ can be thought as a weighted moving average of the past q forecast errors. **Importantly**, *a moving average model must not be confused with moving average smoothing.*
  - The **moving average model** is used for forecasting future values of the time series
  - The **moving average smoothing** is used for noise filtering and for trend estimation.
- Fitting the MA model is **more complicated than the AR model** fitting. This is because the lagged error terms ϵ are not directly observable. Therefore, **iterative methods for non-linear fitting** are applied instead of traditional linear least squares. The description of these methods is not a subject of this material, but can be found here: https://www.it.uu.se/research/publications/reports/2006-022/2006-022-nc.pdf (https://www.it.uu.se/research/publications/reports/2006-022/2006-022-nc.pdf) (Sandgren & Stoica, 2006).

The choice of the maximum lag

Order of the moving average model (maximum lag) can be estimated from the autocorrelation function (not necessarily partial autocorrelation). Similar to the order of the autoregression model, *the maximum lag should be equal to the lag beyond which autocorrelation is equal to zero, typically increased by one.*

Autoregressive moving average (ARMA) model (and its extensions)

The **ARMA model** (sometime called the Box-Jenkins model), is combination of the autoregression and moving average model:

$$\hat{y}_t = \beta_0 + \sum_{i=1}^{p} \beta_i y_{t-i} + \sum_{i=1}^{q} \theta_i \epsilon_{t-i} + e_t$$

where the notation remains the same as in the case of autoregression and autocorrelation. The application of ARMA model to predict y_t is then expressed as:

$$\hat{y}_t = ARMA(p, q)$$

Importantly, ARMA model, similar to MA models, assumes that the time series is stationary, i.e. its mean and standard deviation don't change very much over time. This assumption typically holds only for a short period of time, after which mean "drifts away". To overcome this limitation, a generalization of the ARMA model, called **ARIMA** has be proposed.

The **ARIMA model** (Autoregressive integrated moving average model) is the generalization of the ARMA model, that accounts for the non-stationarity of the time series. The ARIMA model is formally identical to ARMA, but requires an initial **differencing step** to be applied (one or multiple times) to time series y in order to eliminate the non-stationarity. **"Differencing of the time series"** means calculation of the difference between consecutive observations is computed. Mathematically, this is expressed as:

$$y\prime_t = y_t - y_{t-1}$$

Sometime the second degree differentiation is applied:

$$y\prime\prime_t = y\prime_t - y\prime_{t-1}$$

Application of ARIMA model to prediction the y_t then expressed as:

$$\hat{y}_t = ARIMA(p, d, q)$$

where d indicates order of the differentiation used prior the application of the ARMA model. Therefore ARMA, AR, MA can be thought of as special cases of ARIMA, particularly:

- ARIMA(p, 0, q) model is ARMA(p, q)
- ARIMA(p, 0, 0) model AR(p)
- ARIMA(0, 0, q) model MA(q)
- ARIMA(0, 0, 0) model is a **white noise**
- ARIMA(0, 1, 0) model ($y_t = y_{t-1} + \epsilon_t$) is a random signal, called **random walk**

There are multiple extensions to the ARMA/ARIMA models, such as SARIMA, ARIMAX, etc., which are not in the scope of this material. More information about these can be found, for example here: https://arxiv.org/pdf/1302.6613.pdf (https://arxiv.org/pdf/1302.6613.pdf) (Adhikari & Agrawal, 2013).

EXERCISE 5:

Use ARIMA to forecast the number of passengers in the forthcomming month.

- Use 70% of the data for the "training" of the ARIMA model
- Predict the number of passengers in a single forthcoming month ahead.
- Assess the quality of the predictions across the remaining data

In [32]:
```
#your work here
```

In [ ]:

In [ ]:

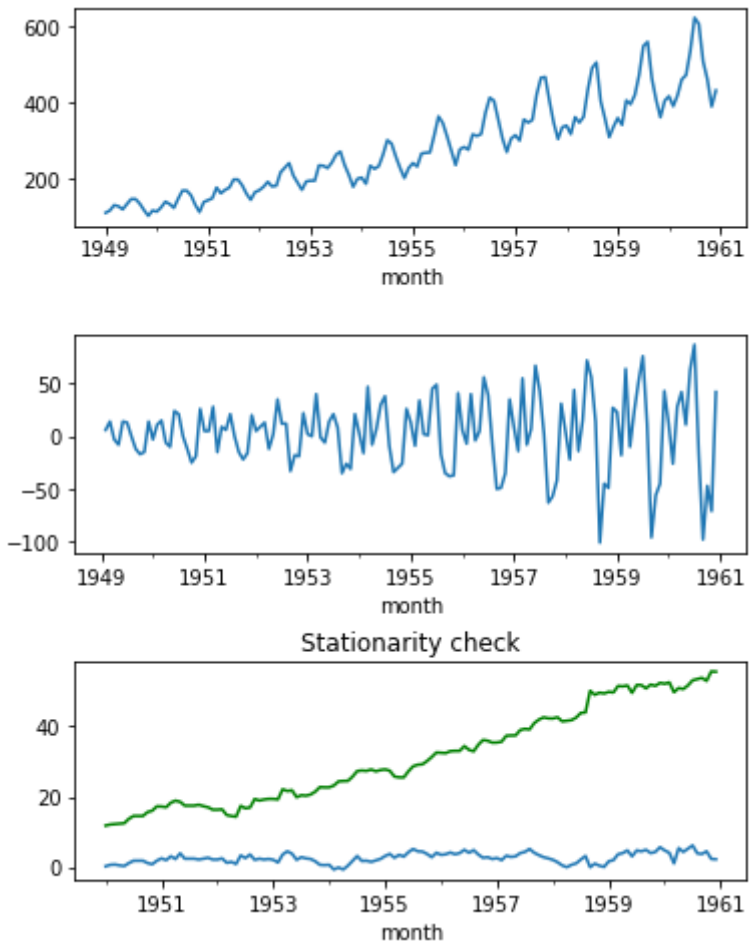
In [ ]:

In [ ]:

Solution

First, we must assess if the assumption of the stationarity of the signal is satisfied. From the previous exercises, we know that signal is not stationary, as there is a strong trend in the signal. To make it stationary we can differentiate the signal and assess the stationarity of the differentiated signal by plotting its moving average and moving standard deviation.

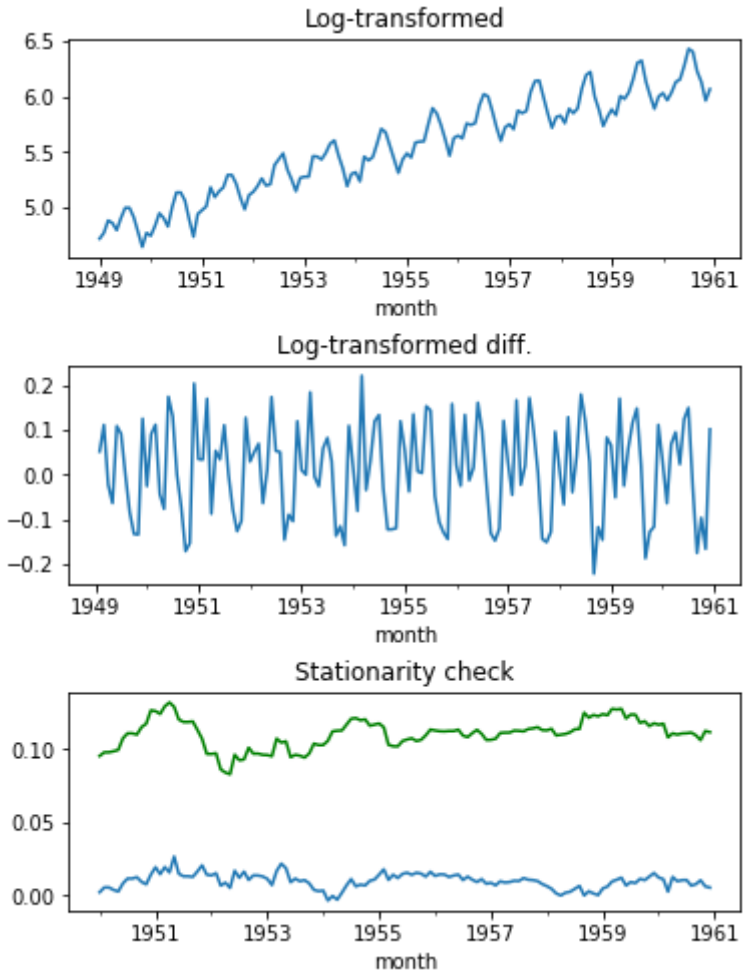
In [33]:
```
fig, (ax1, ax2, ax3) = plt.subplots(3, 1, figsize = (6, 8))

# Plot the passengers numbers
passengers.n_passengers.plot(ax = ax1)

# Plot differentiated signal
passengers.n_passengers.diff().plot(ax = ax2)

# PLot the moving average and moving standard deviation
passengers.n_passengers.diff().rolling(12).mean().plot(ax = ax3, title =
'Stationarity check')
passengers.n_passengers.diff().rolling(12).std().plot(ax = ax3, color =
'green')

# Plot
plt.subplots_adjust(hspace = 0.5)
plt.show()
```



As we can see, even the differentiated signal is not fully stationary as its standard deviation is growing with time. One simple way to resolve it is to logarithmize the signal prior its differentiation:

In [34]:
```
passengers['n_passengers_log'] = np.log(passengers['n_passengers'])
```

In [35]:
```
fig, (ax1, ax2, ax3) = plt.subplots(3, 1, figsize = (6, 8))
# Plot the log-transformed signal
passengers.n_passengers_log.plot(ax = ax1, title = 'Log-transformed')

# Plot differentiated log-transformed signal
passengers.n_passengers_log.diff().plot(ax = ax2, title = 'Log-transform
ed diff.')

# Plot the moving average and moving standard deviation
passengers.n_passengers_log.diff().rolling(12).mean().plot(ax = ax3, tit
le = 'Stationarity check')
passengers.n_passengers_log.diff().rolling(12).std().plot(ax = ax3, colo
r = 'green')
plt.subplots_adjust(hspace = 0.5)
plt.show()
```



You can now see that log-transformed, differentiated signal is stationary.

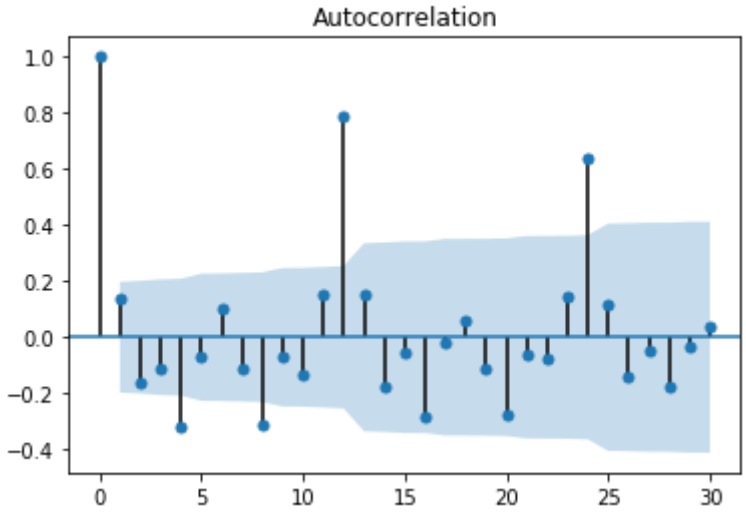
We will split the data to "training" and "validation" set, using approx. 70% of data for training.

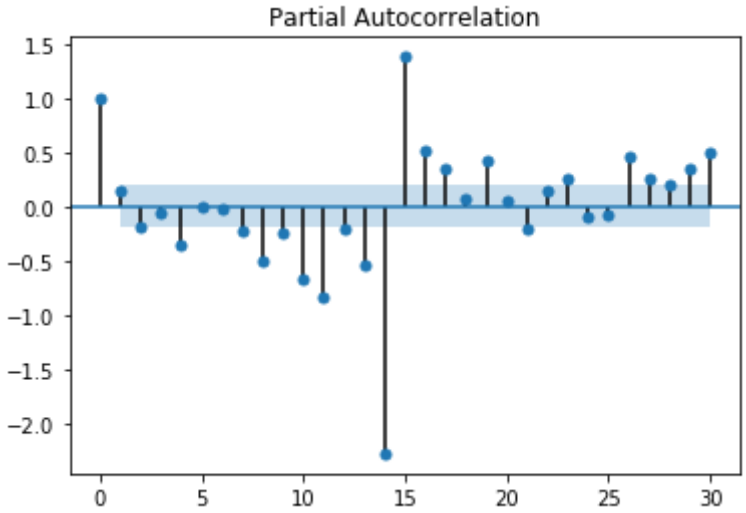
In [36]:
```
n = np.int(len(passengers) * 0.7)
train, test = passengers.iloc[:n], passengers.iloc[n:]
```

Now, we will assess the autocorrelation and partial autocorrelation so that we can set the order of the ARIMA model. Autocorrelations out of the 95% confidence interval will be considered as close to zero.

In [37]:
```
p1 = tsaplots.plot_acf(train.n_passengers_log.diff().dropna(), lags=30,
alpha=0.05)
p2 = tsaplots.plot_pacf(train.n_passengers_log.diff().dropna(), lags=30,
alpha=0.05)
plt.show()
```

```
/anaconda3/lib/python3.7/site-packages/statsmodels/regression/linear_mo
del.py:1283: RuntimeWarning: invalid value encountered in sqrt
  return rho, np.sqrt(sigmasq)
```





As we can see, autocorrelation and partial correlation at the lag = 1 are close to zero. Therefore we may set the order of AR to 1, and order of MA to 1. We will set the order of differentiation to 1, so we work with stationary signal (as shows above). We will use log-transformed data.

In [38]:
```
arima = ARIMA(train.n_passengers_log.dropna().astype(float), order=(1,1,
1))
arima_fit = arima.fit(disp=1, maxiter=100)
print(arima_fit.summary())
```

```
                             ARIMA Model Results
==============================================================================
=======
Dep. Variable:     D.n_passengers_log   No. Observations:
99
Model:                 ARIMA(1, 1, 1)   Log Likelihood
92.419
Method:                       css-mle   S.D. of innovations
0.094
Date:                Fri, 26 Apr 2019   AIC                              -
176.837
Time:                        14:15:54   BIC                              -
166.457
Sample:                    02-01-1949   HQIC                             -
172.637
                         - 04-01-1957
==============================================================================
=====================
                                 coef    std err          z      P>|z|
[0.025      0.975]
------------------------------------------------------------------------------
---------------------
const                          0.0110      0.001     11.822      0.000
0.009       0.013
ar.L1.D.n_passengers_log       0.6729      0.076      8.814      0.000
0.523       0.822
ma.L1.D.n_passengers_log      -1.0000      0.026    -38.736      0.000
-1.051      -0.949
                                    Roots
==============================================================================
======
                  Real          Imaginary           Modulus         Fre
quency
------------------------------------------------------------------------------
------
AR.1            1.4862           +0.0000j            1.4862
0.0000
MA.1            1.0000           +0.0000j            1.0000
0.0000
------------------------------------------------------------------------------
------

/anaconda3/lib/python3.7/site-packages/statsmodels/tsa/base/tsa_model.p
y:171: ValueWarning: No frequency information was provided, so inferred
frequency MS will be used.
  % freq, ValueWarning)
/anaconda3/lib/python3.7/site-packages/statsmodels/tsa/base/tsa_model.p
y:171: ValueWarning: No frequency information was provided, so inferred
frequency MS will be used.
  % freq, ValueWarning)
```

Notice that the summary of the model is similar to the summary of the regression model described in the previous modules. For more details, please inspect the documentation of the *statsmodels* library: https://devdocs.io/statsmodels/ (https://devdocs.io/statsmodels/) (Statsmodel developers and others, 2012).

We will forecast the number of passengers in the forthcoming month.

In [39]:
```
p, err, conf = arima_fit.forecast()
print ('Predicted no. of passengers [log]: %1.2f' %  p)
print ('Standard error of the prediction: %1.2f' % err)
print ('COnfidence interval: (%1.2f, %1.2f)' % (conf[0][0], conf[0][1]))

y_hat = np.exp(p)
y = test.n_passengers[0]
```

```
Predicted no. of passengers [log]: 5.87
Standard error of the prediction: 0.09
COnfidence interval: (5.69, 6.05)
```

Now we will compare the predicted values with actual values

In [40]:
```
print ('Predicted number of pasengers: %1.2f' % y_hat)
print ('Actual number of passengers: %i' % y)
print ('Absolute error: %1.2f' % np.abs(y_hat - y))
print ('Relative error %1.2f%%' % (np.abs(y_hat - y) / y * 100))
```

```
Predicted number of pasengers: 354.46
Actual number of passengers: 355
Absolute error: 0.54
Relative error 0.15%
```

As we can see, the predicted value is quite close to the actual one. Let's extend our validation to other months. We will do iterative **in-sample** forecasting. The history is initially formed of the first 70% of a time series. Then we will forecast the first point of the test set, and will add the true value to the history. Then we will forecast the second point etc. This will provide us an evaluation of the model's predictive quality.

In [41]:
```
# Define the history
hist = list(train.n_passengers_log.dropna().astype(float))

# Initialize container for predictions
pred = []
for i in range(len(test)):
    # Initalize the model
    arima = ARIMA(hist, order=(6,1,0))
    #Fit the model on the available history
    arima_fit = arima.fit(disp=1, maxiter=500)
    # Take the prediction
    pred.append(arima_fit.forecast()[0][0])
    # Expand the history
    hist.append(test.n_passengers_log[i])
```

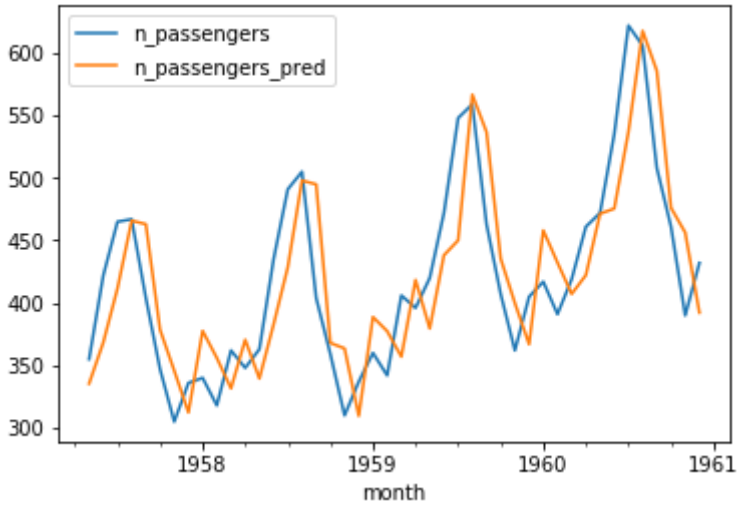
```
/anaconda3/lib/python3.7/site-packages/scipy/signal/signaltools.py:134
1: FutureWarning: Using a non-tuple sequence for multidimensional index
ing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the
future this will be interpreted as an array index, `arr[np.array(seq)]
`, which will result either in an error or a different result.
  out_full[ind] += zi
/anaconda3/lib/python3.7/site-packages/scipy/signal/signaltools.py:134
4: FutureWarning: Using a non-tuple sequence for multidimensional index
ing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the
future this will be interpreted as an array index, `arr[np.array(seq)]
`, which will result either in an error or a different result.
  out = out_full[ind]
/anaconda3/lib/python3.7/site-packages/scipy/signal/signaltools.py:135
0: FutureWarning: Using a non-tuple sequence for multidimensional index
ing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the
future this will be interpreted as an array index, `arr[np.array(seq)]
`, which will result either in an error or a different result.
  zf = out_full[ind]
```

Now we will compare the predictions with the actual values.

In [42]:
```
test['n_passengers_pred'] = np.exp(pred)
test[['n_passengers', 'n_passengers_pred']].plot()
```

```
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:1: Setting
WithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-d
ocs/stable/indexing.html#indexing-view-versus-copy
  """Entry point for launching an IPython kernel.
```

Out[42]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1c22f626a0>
```

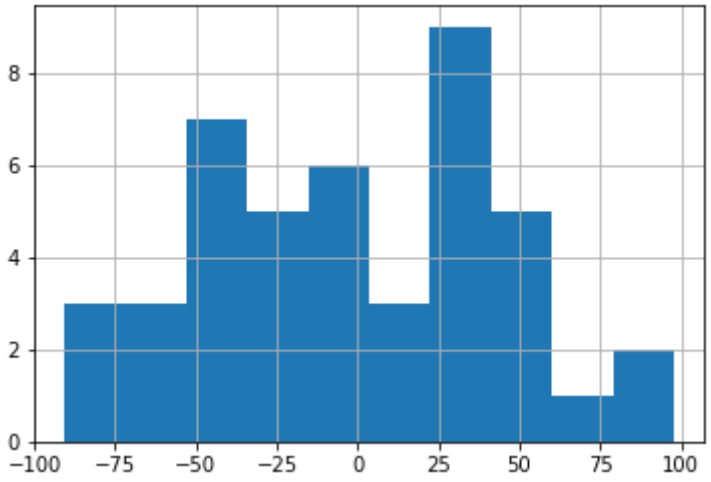


As we can see, the predictions follow the actual values quite nicely. To get a more quantitative estimate of how good the predictions are, we plot the histogram of the error, and can calculate the root mean square error.

In [43]:
```
test['error'] = test.n_passengers - test.n_passengers_pred
test.error.hist()
```

```
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:1: Setting
WithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-d
ocs/stable/indexing.html#indexing-view-versus-copy
  """Entry point for launching an IPython kernel.
```

Out[43]: <matplotlib.axes._subplots.AxesSubplot at 0x1c247cc2b0>



In [44]:
```
rmse = np.sqrt(np.mean(np.square(test.error - np.mean(test.error))))
print (rmse)
```

```
45.490817863921094
```

Long-term Predictions

Sometimes, we need to forecast more than just one time step ahead. This is can be addressed in two ways:

- **A direct forecast**. This forecast model is designed in a way that it directly estimates the value of y in a more distant future. For example, in case of the autoregression model it can be formulated as:

$$\hat{y}_{t+h} = \beta_0 + \sum_{i=0}^{p} \beta_i \, y_t + e_{t+h}$$

  where h - is called **prediction horizon**.

- **An iterative (inductive) forecast**. This means we will do a sequence of short-term predictions (one step ahead), where each prediction uses previous prediction estimates as if it would be an actual value.

Word of caution with long-term predictions

Regardless of whether iterative, or direct forecast is used, the **predictability** of future values of any time-series decreases rapidly with increases of the prediction horizon. How far into the future we can predict the time series values, depends on many factors, most prominently:

- Amount and frequency of observations
- Strength of seasonality, trend and noisiness
- Forecasting method
- Domain specific factors

There are multiple methods for estimating the maximum prediction horizon. For more details see for example: https://robjhyndman.com/papers/ijf25.pdf (https://robjhyndman.com/papers/ijf25.pdf) (De Gooijer & Hyndman, 2006).

The simplest possible (and often misleading) rule is:

$$h_{max} \sim \frac{1}{6} \cdot \frac{N}{F} \cdot \frac{T}{\Delta t}$$

where:

- N is the number of historical observations we dispose by to assess the prediction performance of the forecasting method.
- F is the frequency, i.e. number of observations within a single period/season.
- Δt is the length of the time step and T is the length of the period.

However, **long-term predictions should be avoided unless absolutely necessary**.

"*In a long run we are all dead*" -- Keynes

EXERCISE 6:

Use the ARIMA model to forecast the number of passengers across the whole testing set (from the previous exercises). Restrict your analysis to use only the training set data when fitting the model, and use iterative forecasting. See what is the maximum prediction horizon for which the predictions hold. Adjust the order of the model, see if the predictions improve.

In [45]:
```
#your work here
```

In [ ]:

In [ ]:

In [ ]:

Solution

Let's fit the model first, using the training data only.

In [46]:
```
arima = ARIMA(train.n_passengers_log.dropna().astype(float), order=(1,1,
1))
arima_fit = arima.fit(disp=1, maxiter=100)
```

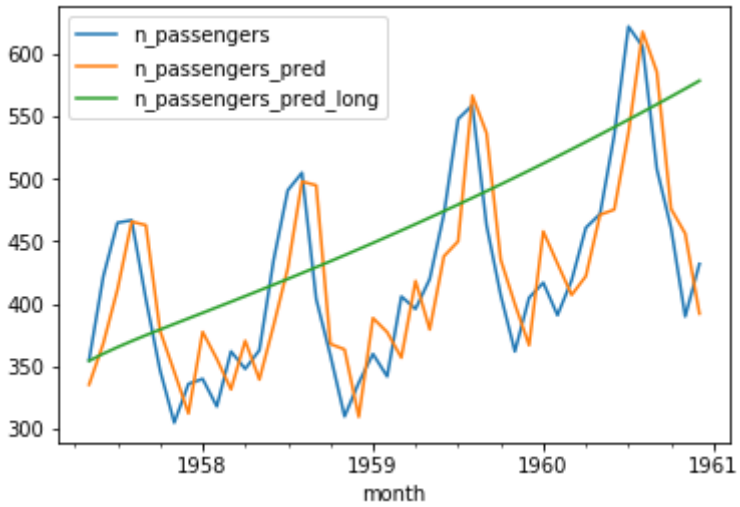
```
/anaconda3/lib/python3.7/site-packages/statsmodels/tsa/base/tsa_model.p
y:171: ValueWarning: No frequency information was provided, so inferred
frequency MS will be used.
  % freq, ValueWarning)
/anaconda3/lib/python3.7/site-packages/statsmodels/tsa/base/tsa_model.p
y:171: ValueWarning: No frequency information was provided, so inferred
frequency MS will be used.
  % freq, ValueWarning)
```

Now we will forecast across the whole testing set. We will do out-of-sample forecasting by iteratively predicting the next point of the test series, and appending this prediction to the history. This approach for long-term forecasting is implemented in the *forecast* function.

In [47]:
```
pred,_,_ = arima_fit.forecast(steps=len(test))
test['n_passengers_pred_long'] = np.exp(pred)
test[['n_passengers', 'n_passengers_pred', 'n_passengers_pred_long']].pl
ot()
```

```
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:2: Setting
WithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-d
ocs/stable/indexing.html#indexing-view-versus-copy
```

Out[47]: <matplotlib.axes._subplots.AxesSubplot at 0x1c246c2358>



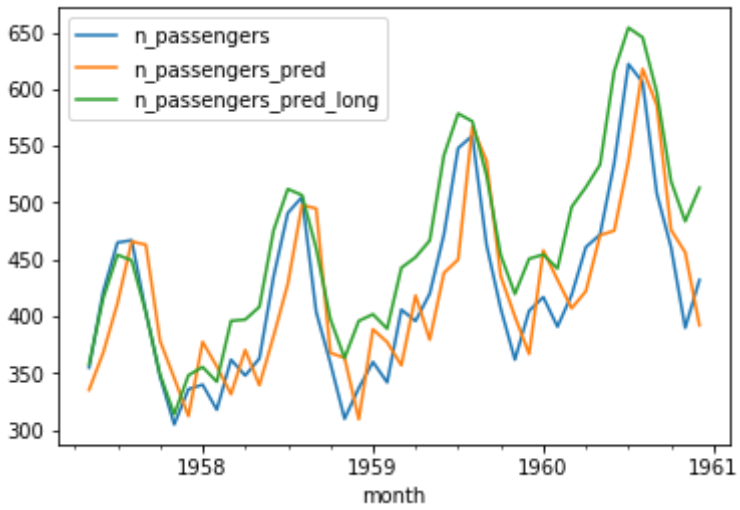
As we can see, the predictions follow the overall trend, but are quite off.

Let's increase the order of the model, so that the order of AR will be equal to the period of the signal (12) and order of the MA will be equal to 5, which is one lag beyond the next lag where the autocorrelation of the signal is significant.

In [48]:
```
arima = ARIMA(train.n_passengers_log.dropna().astype(float), order=(12,1
,6))
arima_fit = arima.fit(disp=1, maxiter=500)
pred,_,_ = arima_fit.forecast(steps=len(test))
test['n_passengers_pred_long'] = np.exp(pred)
test[['n_passengers', 'n_passengers_pred', 'n_passengers_pred_long']].pl
ot()
```

```
/anaconda3/lib/python3.7/site-packages/statsmodels/tsa/base/tsa_model.p
y:171: ValueWarning: No frequency information was provided, so inferred
frequency MS will be used.
  % freq, ValueWarning)
/anaconda3/lib/python3.7/site-packages/statsmodels/tsa/base/tsa_model.p
y:171: ValueWarning: No frequency information was provided, so inferred
frequency MS will be used.
  % freq, ValueWarning)
/anaconda3/lib/python3.7/site-packages/statsmodels/base/model.py:508: C
onvergenceWarning: Maximum Likelihood optimization failed to converge.
Check mle_retvals
  "Check mle_retvals", ConvergenceWarning)
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:4: Setting
WithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-d
ocs/stable/indexing.html#indexing-view-versus-copy
  after removing the cwd from sys.path.
```

Out[48]: <matplotlib.axes._subplots.AxesSubplot at 0x1c24363160>



As we can see, the prediction are much better, but still worse that the in-sample prediction from the previous exercise. By increasing the order of our model, the model takes into account more of the signal past. It also added more "flexibility" to the model, so it can now fit the signal better. Feel free to adjust the order of the model to obtain the best predictions.

End of Module.

You have reached the end of this module. If you have any questions, please reach out to your peers using the discussion boards.

If you and your peers are unable to come to a suitable conclusion, do not hesitate to reach out to your instructor on the designated discussion board.

When you are comfortable with the content, and have practiced to your satisfaction, you may proceed to any related assignments, and to the next module.

References

- Adhikari, R., & Agrawal, R.K. (2013). An Introductory Study on Time Series Modeling and Forecasting. Lambert Academic Publishing, Germany. Retrieved Dec 14, 2018 from https://arxiv.org/pdf/1302.6613.pdf (https://arxiv.org/pdf/1302.6613.pdf)
- De Gooijer, J.G. & Hyndman, R.J. (2006). 25 years of time series forecasting. International Journal of Forecasting, 22(3), 443-473. Retrieved on Oct 25, 2018 from https://robjhyndman.com/papers/ijf25.pdf (https://robjhyndman.com/papers/ijf25.pdf)
- Moving Average, (n.d). Retrieved from Wikipedia on Oct 25, 2018 from https://en.wikipedia.org/wiki/Moving_average (https://en.wikipedia.org/wiki/Moving_average)
- Sandgren, N., & Stoica, P. (2006). On Moving Average Parameter Estimation. Department of Information Technology, Uppsala University. Retrieved on Oct 25, 2018 from https://www.it.uu.se/research/publications/reports/2006-022/2006-022-nc.pdf (https://www.it.uu.se/research/publications/reports/2006-022/2006-022-nc.pdf)
- Statsmodel developers and others (2012). Statsmodels. Retrieved Dec 14, 2018 from https://devdocs.io/statsmodels/ (https://devdocs.io/statsmodels/)