Advanced Time Series Modeling and Forecasting

Neba Nfonsang
University of Denver

In [1]:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pydataset
import pandas_datareader as pdr
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.tsa.stattools import acf
from pandas.plotting import autocorrelation_plot
from statsmodels.formula.api import ols
from statsmodels.tsa.ar_model import AR
from statsmodels.tsa.seasonal import seasonal_decompose
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from statsmodels.tsa.arima_model import ARMA
from statsmodels.tsa.arima_model import ARIMA
import statsmodels.tsa
```

In [2]:

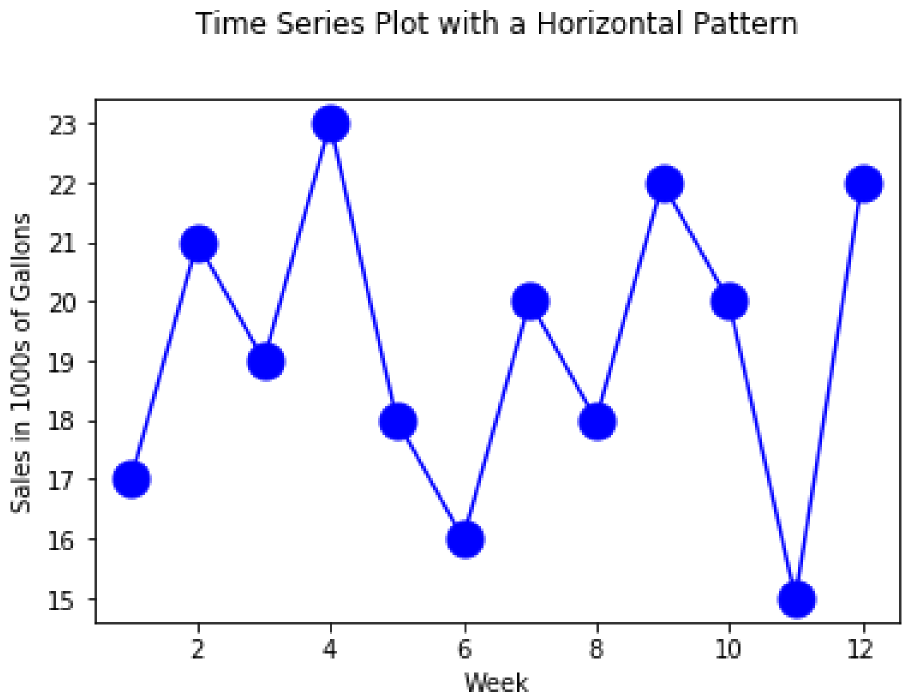
```
file = r"C:\Users\nnfon\Desktop\DS_Tools_1_Winter_2020\Data\time_ser
gasoline = pd.read_excel(file)
gasoline = gasoline.rename(columns={'Sales (1000s of gallons)':"Sale
gasoline.head()
```

Out[2]:

	Week	Sales
0	1	17
1	2	21
2	3	19
3	4	23
4	5	18

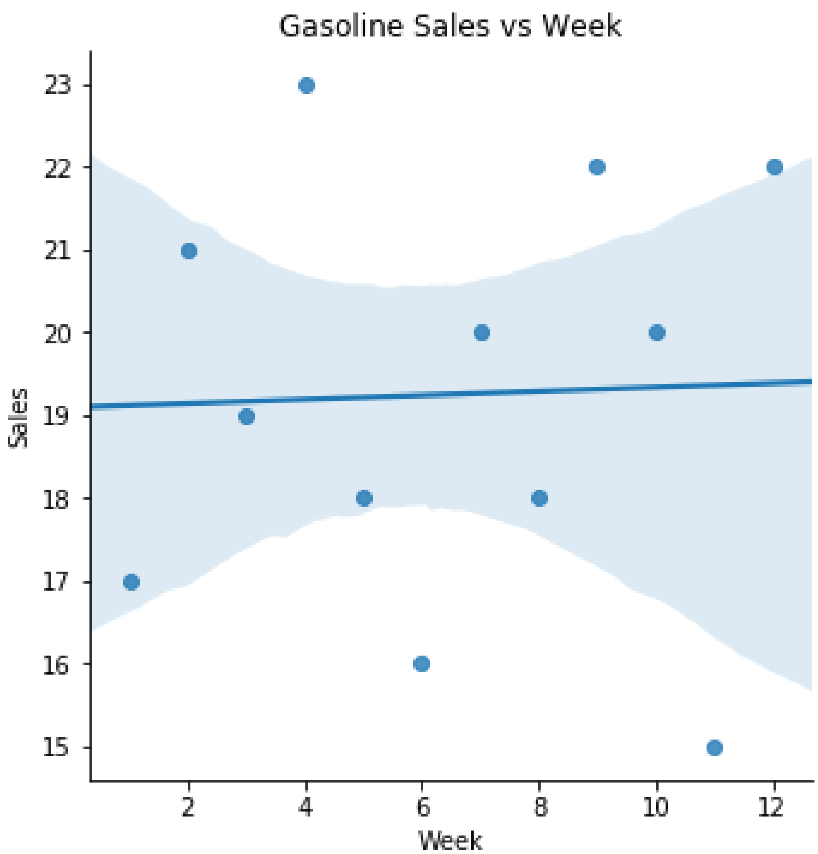
In [3]:

```
plt.plot(gasoline.Week, gasoline.Sales, marker="o", markersize=15, c
plt.title("Time Series Plot with a Horizontal Pattern", y=1.1)
plt.xlabel("Week")
plt.ylabel("Sales in 1000s of Gallons")
plt.show()
```



In [4]:

```
sns.lmplot(x="Week", y="Sales", data=gasoline)
plt.title("Gasoline Sales vs Week", y=1.1);
```



In [5]:

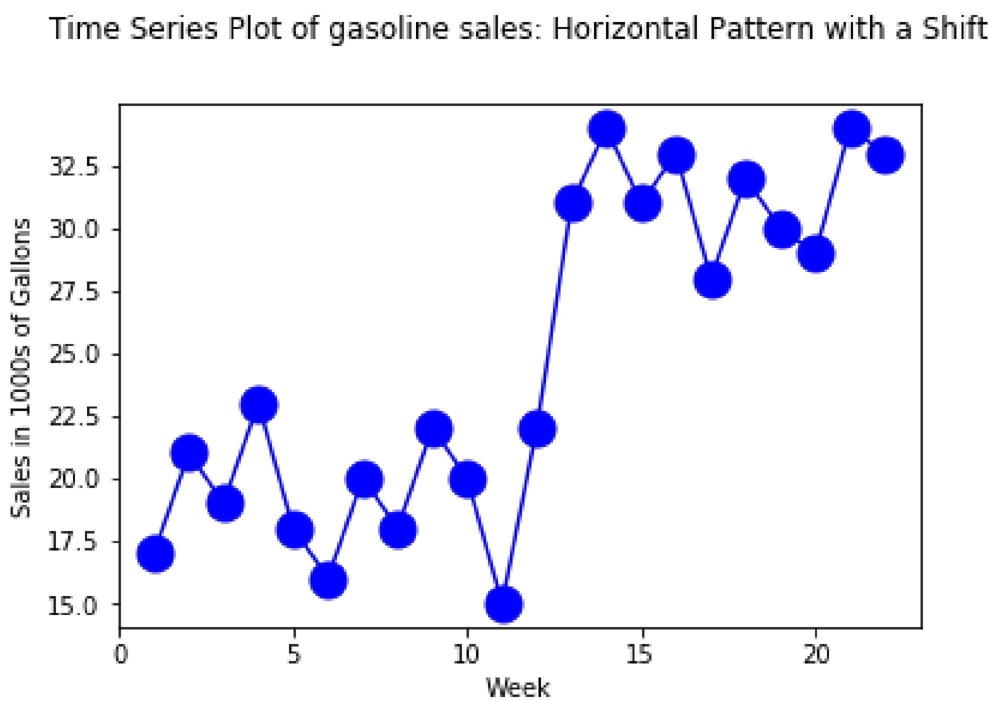
```
file = r"C:\Users\nnfon\Desktop\DS_Tools_1_Winter_2020\Data\time_ser
gasoline_shift = pd.read_excel(file)
gasoline_shift = gasoline_shift.rename(columns={'Sales (1000s of gal
gasoline_shift.head()
```

Out[5]:

	Week	Sales
0	1	17
1	2	21
2	3	19
3	4	23
4	5	18

In [6]:

```
plt.plot(gasoline_shift.Week, gasoline_shift.Sales, marker="o", mark
plt.title("Time Series Plot of gasoline sales: Horizontal Pattern wi
plt.xlabel("Week")
plt.ylabel("Sales in 1000s of Gallons")
plt.show()
```



In [7]:

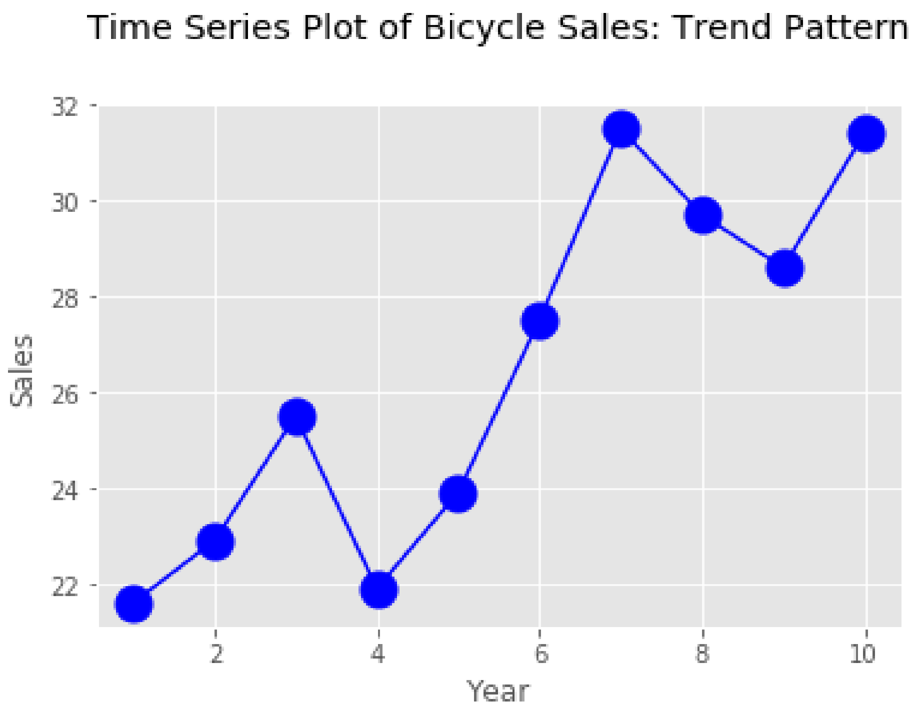
```
bike = r"C:\Users\nnfon\Desktop\DS_Tools_1_Winter_2020\Data\time_ser
bicycle = pd.read_excel(bike)
bicycle
```

Out[7]:

	Year	Sales
0	1	21.6
1	2	22.9
2	3	25.5
3	4	21.9
4	5	23.9
5	6	27.5
6	7	31.5
7	8	29.7
8	9	28.6
9	10	31.4

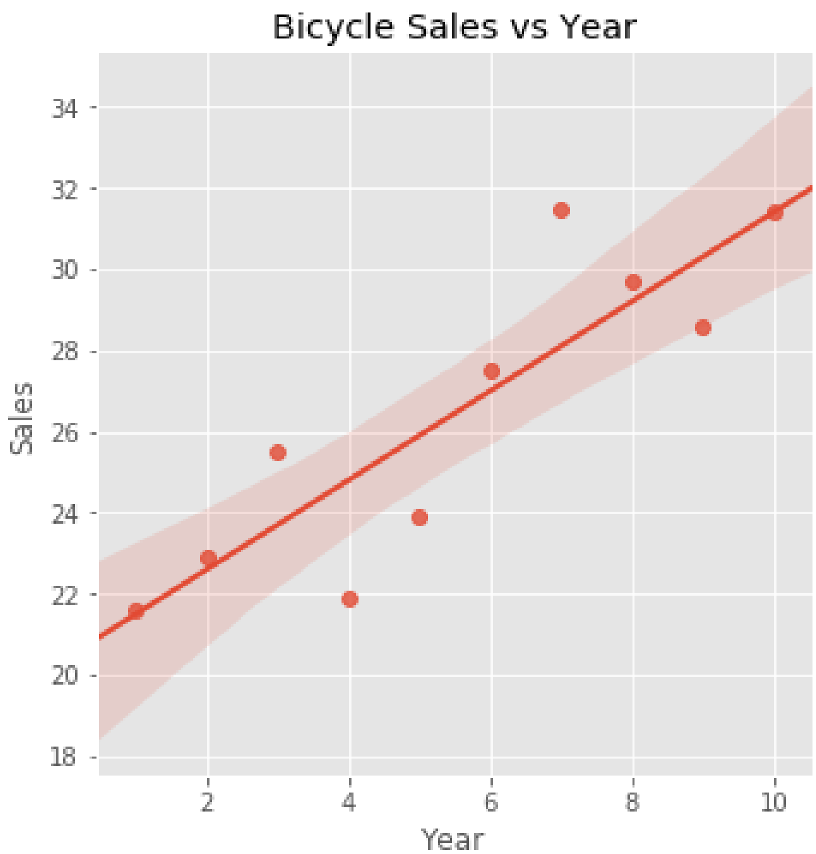
In [8]:

```
plt.style.use("ggplot")
plt.plot(bicycle.Year, bicycle.Sales, marker="o", markersize=15, col
plt.title("Time Series Plot of Bicycle Sales: Trend Pattern", y=1.1)
plt.xlabel("Year")
plt.ylabel("Sales")
plt.show()
```



In [9]:

```
sns.lmplot(x="Year", y="Sales", data=bicycle)
plt.title("Bicycle Sales vs Year", y=1.1);
```



In [10]:

```
chol = r"C:\Users\nnfon\Desktop\DS_Tools_1_Winter_2020\Data\time_ser
cholesterol = pd.read_excel(chol)
cholesterol
```

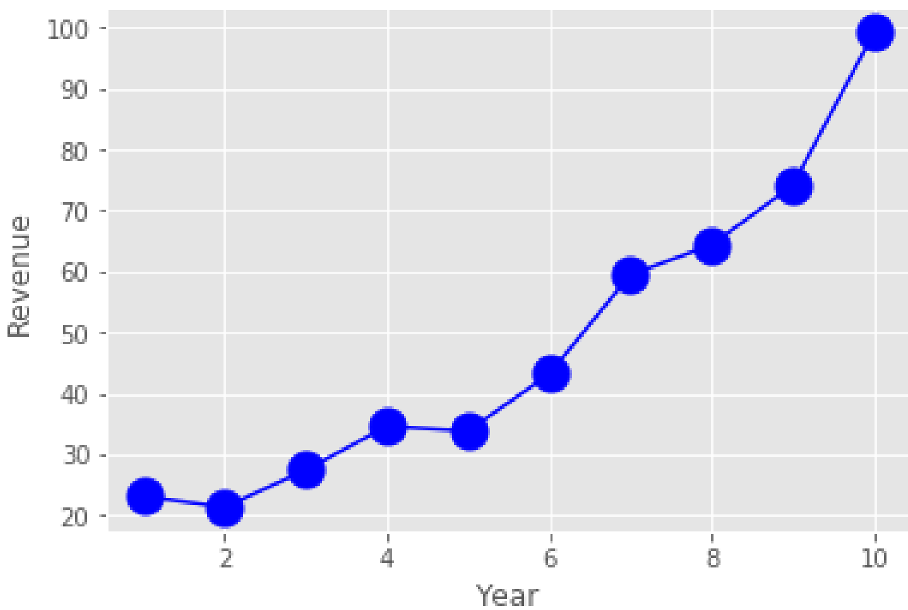
Out[10]:

	Year	Revenue
0	1	23.1
1	2	21.3
2	3	27.4
3	4	34.6
4	5	33.8
5	6	43.2
6	7	59.5
7	8	64.4
8	9	74.2
9	10	99.3

In [11]:

```
plt.style.use("ggplot")
plt.plot(cholesterol.Year, cholesterol.Revenue, marker="o", markersi
plt.title("Time Series Plot of Cholesterol Drug Revenue: Seasonal Pa
plt.xlabel("Year")
plt.ylabel("Revenue")
plt.show()
```

Time Series Plot of Cholesterol Drug Revenue: Seasonal Pattern



In [12]:

```
umbr = r"C:\\Users\nnfon\\Desktop\\DS_Tools_1_Winter_2020\\Data\time
umbrella = pd.read_excel(umbr)
umbrella["Year"] = [1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 4, 4, 4, 4,

print(umbrella.shape)
umbrella.head(10)
```

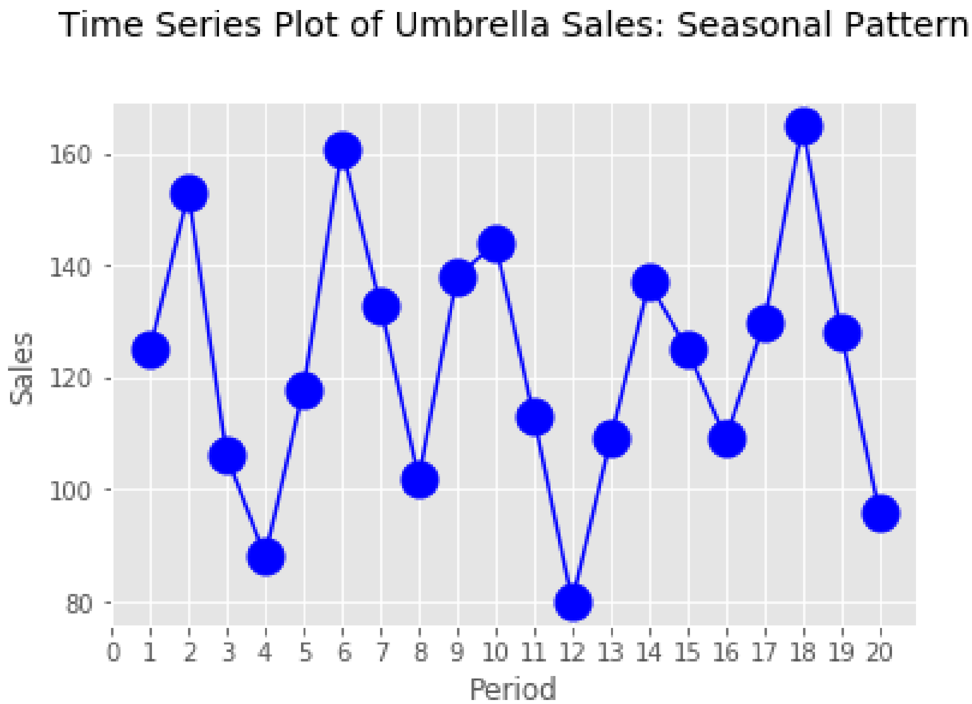
```
(20, 4)
```

Out[12]:

	Year	Quarter	Period	Sales
0	1	1	1	125
1	1	2	2	153
2	1	3	3	106
3	1	4	4	88
4	2	1	5	118
5	2	2	6	161
6	2	3	7	133
7	2	4	8	102
8	3	1	9	138
9	3	2	10	144

In [13]:

```
plt.style.use("ggplot")
plt.plot(umbrella.Period, umbrella.Sales, marker="o", markersize=15,
plt.title("Time Series Plot of Umbrella Sales: Seasonal Pattern", y=
plt.xlabel("Period")
plt.ylabel("Sales")
plt.xticks(range(21))
plt.show()
```



In [14]:

```
phone = r"C:\\Users\nnfon\\Desktop\\DS_Tools_1_Winter_2020\\Data\tim
phone = pd.read_excel(phone)

print(phone.shape)
phone.head(10)
```

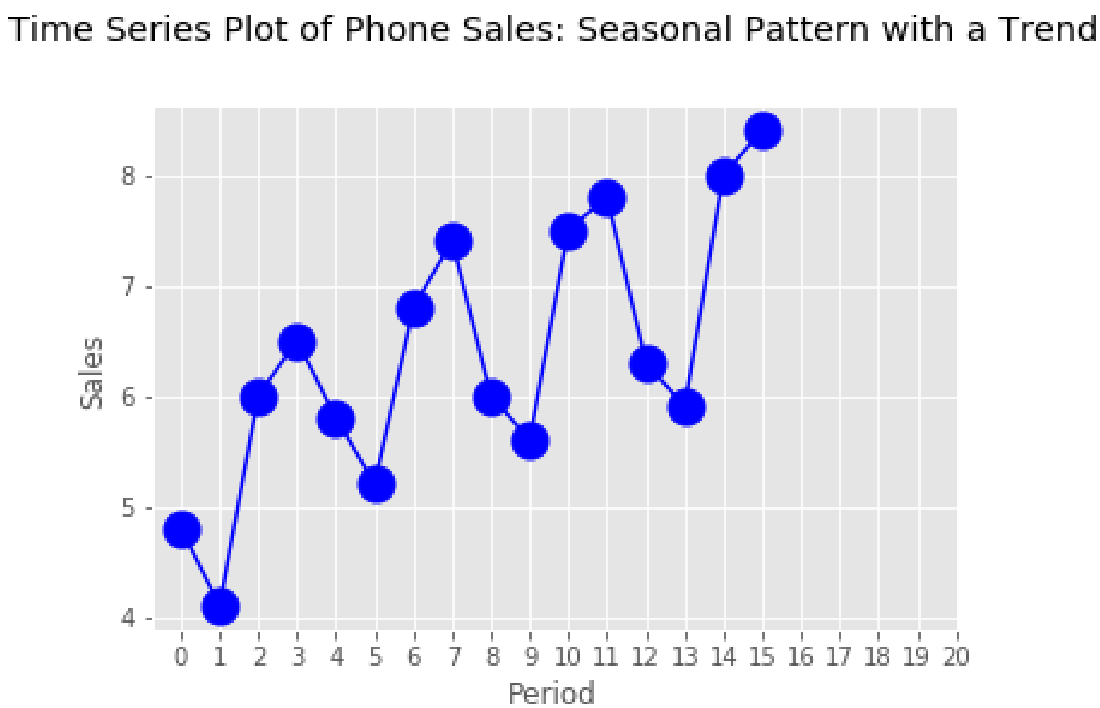
```
(16, 3)
```

Out[14]:

	Year	Quarter	Sales (1000s)
0	1	1	4.8
1	1	2	4.1
2	1	3	6.0
3	1	4	6.5
4	2	1	5.8
5	2	2	5.2
6	2	3	6.8
7	2	4	7.4
8	3	1	6.0
9	3	2	5.6

In [15]:

```
plt.style.use("ggplot")
plt.plot(phone.index, phone["Sales (1000s)"], marker="o", markersize
plt.title("Time Series Plot of Phone Sales: Seasonal Pattern with a 
plt.xlabel("Period")
plt.ylabel("Sales")
plt.xticks(range(21))
plt.show()
```



In [16]:

```
import pydataset
```

In [17]:

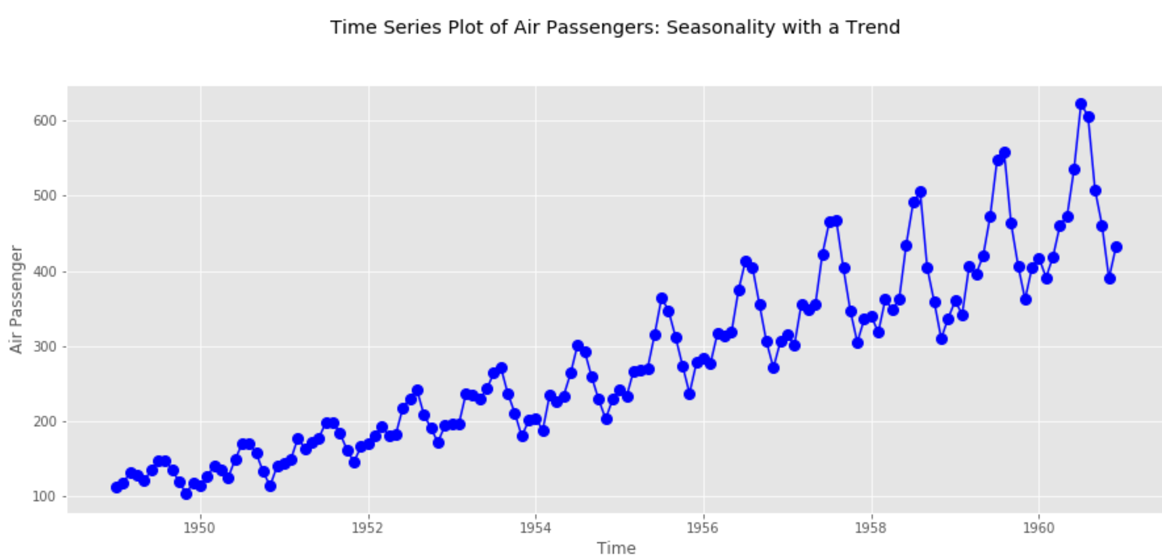
```
air_passenger = pydataset.data("AirPassengers")
air_passenger.head()
```

Out[17]:

	time	AirPassengers
1	1949.000000	112
2	1949.083333	118
3	1949.166667	132
4	1949.250000	129
5	1949.333333	121

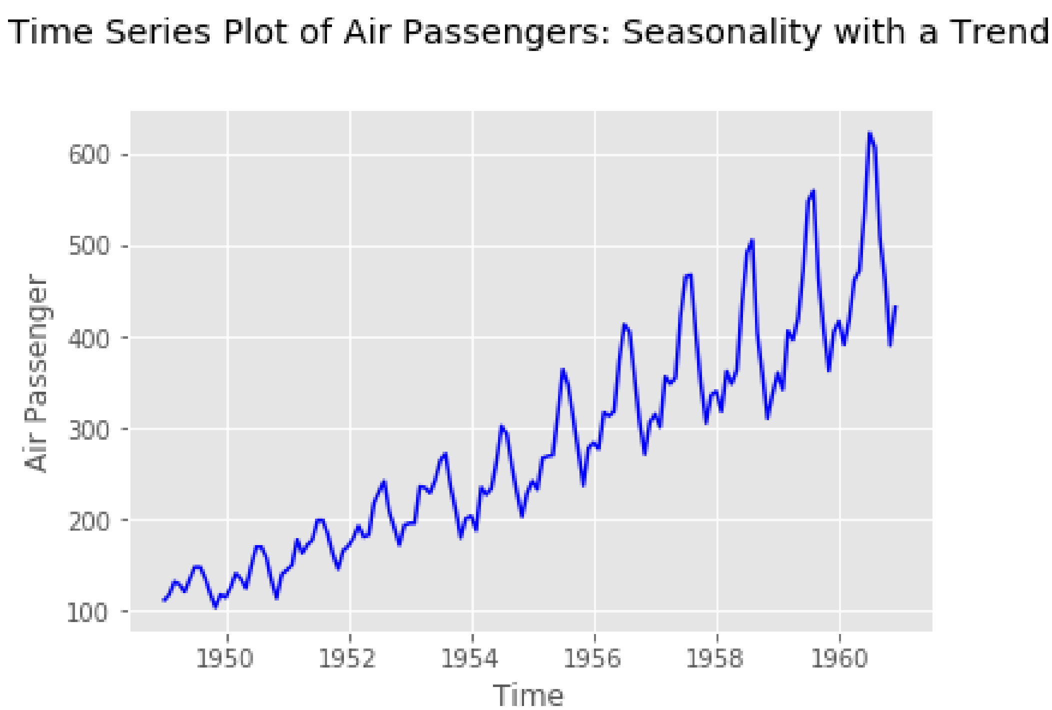
In [18]:

```
plt.style.use("ggplot")
plt.figure(figsize=(15, 6))
plt.title("Time Series Plot of Air Passengers: Seasonality with a Tr
plt.xlabel("Time")
plt.ylabel("Air Passenger")
plt.plot(air_passenger.time,
         air_passenger.AirPassengers, c="b", marker="o", markersize=
```



In [19]:

```
plt.style.use("ggplot")
#plt.figure(figsize=(15, 6))
plt.title("Time Series Plot of Air Passengers: \
Seasonality with a Trend", y=1.1)
plt.xlabel("Time")
plt.ylabel("Air Passenger")
plt.plot(air_passenger.time,
         air_passenger.AirPassengers, c="b");
```



In [20]:

```
# to apply rolling window,
# first set the time as index
air_pass = air_passenger.set_index("time")
air_pass.head()
```

Out[20]:

	AirPassengers
time	
1949.000000	112
1949.083333	118
1949.166667	132
1949.250000	129
1949.333333	121

In [21]:

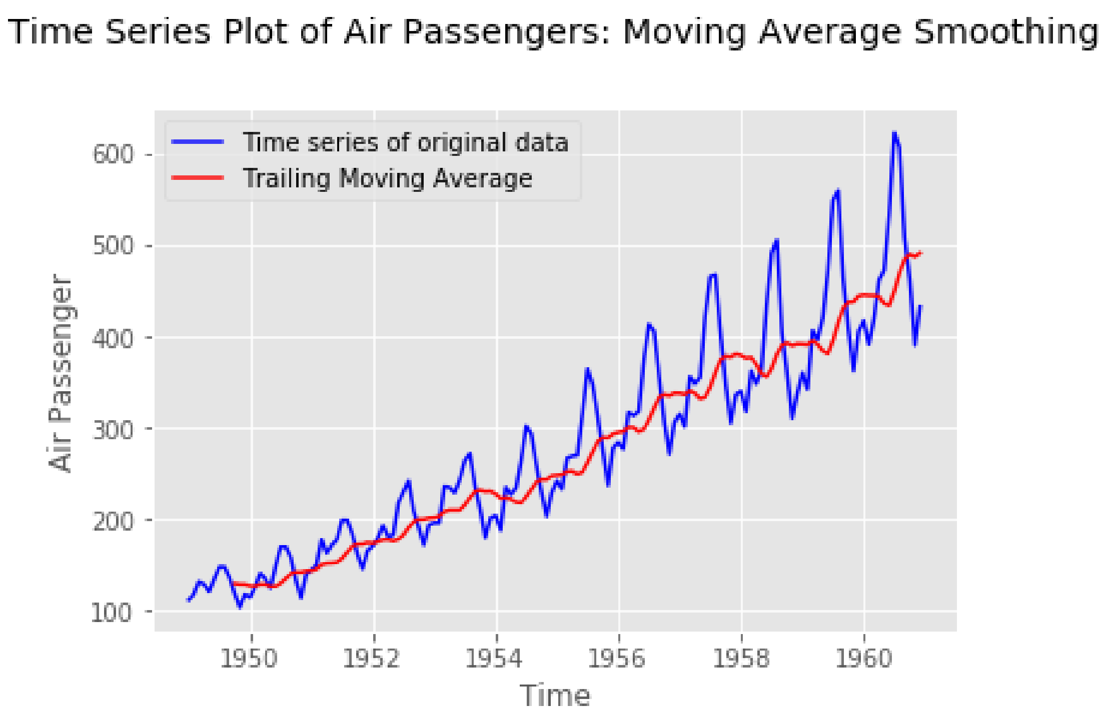
```
rolling = air_pass.rolling(10).mean()
rolling.head(12)
```

Out[21]:

	AirPassengers
time	
1949.000000	NaN
1949.083333	NaN
1949.166667	NaN
1949.250000	NaN
1949.333333	NaN
1949.416667	NaN
1949.500000	NaN
1949.583333	NaN
1949.666667	NaN
1949.750000	129.8
1949.833333	129.0
1949.916667	129.0

In [22]:

```
plt.style.use("ggplot")
#plt.figure(figsize=(15, 6))
plt.title("Time Series Plot of Air Passengers: \
Moving Average Smoothing", y=1.1)
plt.xlabel("Time")
plt.ylabel("Air Passenger")
plt.plot(air_passenger.time,
         air_passenger.AirPassengers, c="b")
plt.plot(rolling.index,
         rolling.AirPassengers, c="r")
plt.legend(["Time series of original data",
            "Trailing Moving Average"]);
```



In [23]:

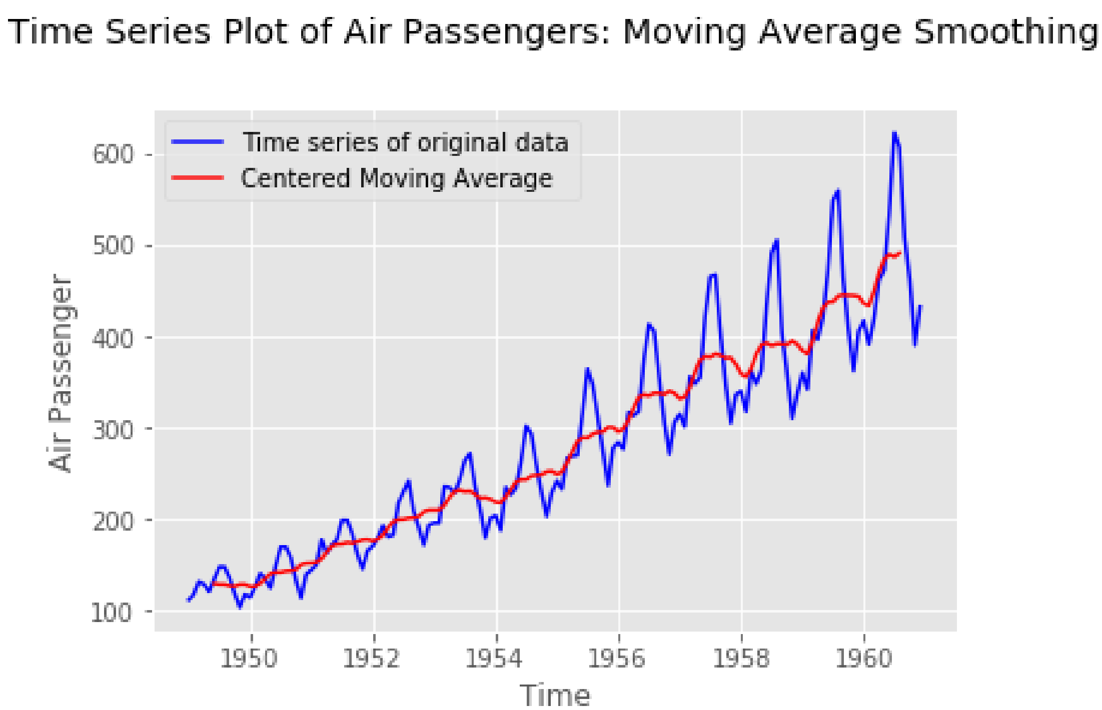
```
# centered moving average
rolling_centered = air_pass.rolling(10, center=True).mean()
rolling_centered.head(12)
```

Out[23]:

	AirPassengers
time	
1949.000000	NaN
1949.083333	NaN
1949.166667	NaN
1949.250000	NaN
1949.333333	NaN
1949.416667	129.8
1949.500000	129.0
1949.583333	129.0
1949.666667	127.3
1949.750000	127.0
1949.833333	129.0
1949.916667	129.0

In [24]:

```
plt.style.use("ggplot")
#plt.figure(figsize=(15, 6))
plt.title("Time Series Plot of Air Passengers: \
Moving Average Smoothing", y=1.1)
plt.xlabel("Time")
plt.ylabel("Air Passenger")
plt.plot(air_passenger.time,
         air_passenger.AirPassengers, c="b")
plt.plot(rolling_centered.index,
         rolling_centered.AirPassengers, c="r")
plt.legend(["Time series of original data",
            "Centered Moving Average"]);
```



In [25]:

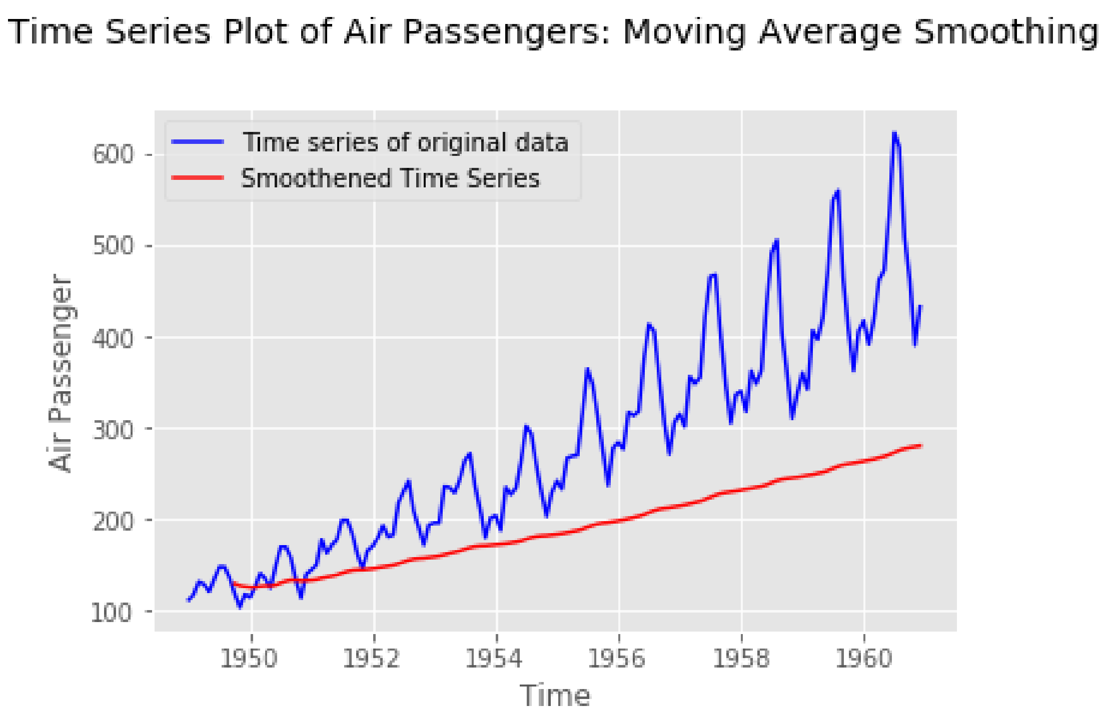
```
expanding = air_pass.expanding(10).mean()
expanding.head(15)
```

Out[25]:

	AirPassengers
time	
1949.000000	NaN
1949.083333	NaN
1949.166667	NaN
1949.250000	NaN
1949.333333	NaN
1949.416667	NaN
1949.500000	NaN
1949.583333	NaN
1949.666667	NaN
1949.750000	129.800000
1949.833333	127.454545
1949.916667	126.666667
1950.000000	125.769231
1950.083333	125.785714
1950.166667	126.800000

In [26]:

```
plt.style.use("ggplot")
#plt.figure(figsize=(15, 6))
plt.title("Time Series Plot of Air Passengers: \
Moving Average Smoothing", y=1.1)
plt.xlabel("Time")
plt.ylabel("Air Passenger")
plt.plot(air_passenger.time,
         air_passenger.AirPassengers, c="b")
plt.plot(expanding.index,
         expanding.AirPassengers, c="r")
plt.legend(["Time series of original data",
            "Smoothened Time Series"]);
```



In [27]:

```
# let say you wanted to shift a time series backward by a
# lag of 12
expanding.shift(-12).head()
```

Out[27]:

	AirPassengers
time	
1949.000000	125.769231
1949.083333	125.785714
1949.166667	126.800000
1949.250000	127.312500
1949.333333	127.176471

In [28]:

```
# view
air_pass.head()
```

Out[28]:

	AirPassengers
time	
1949.000000	112
1949.083333	118
1949.166667	132
1949.250000	129
1949.333333	121

In [29]:

```
air = air_pass.copy()
air["lag1"] = air_pass["AirPassengers"].shift(1)
air.head()
```

Out[29]:

	AirPassengers	lag1
time		
1949.000000	112	NaN
1949.083333	118	112.0
1949.166667	132	118.0
1949.250000	129	132.0
1949.333333	121	129.0

In [30]:

```
for i in range(1, 21):
    air[f"lag{i}"] = air_pass["AirPassengers"].shift(i)
air.head(10)
```

Out[30]:

	AirPassengers	lag1	lag2	lag3	lag4	lag5	lag6	lag7	la
time									
1949.000000	112	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
1949.083333	118	112.0	NaN	NaN	NaN	NaN	NaN	NaN	N
1949.166667	132	118.0	112.0	NaN	NaN	NaN	NaN	NaN	N
1949.250000	129	132.0	118.0	112.0	NaN	NaN	NaN	NaN	N
1949.333333	121	129.0	132.0	118.0	112.0	NaN	NaN	NaN	N
1949.416667	135	121.0	129.0	132.0	118.0	112.0	NaN	NaN	N
1949.500000	148	135.0	121.0	129.0	132.0	118.0	112.0	NaN	N
1949.583333	148	148.0	135.0	121.0	129.0	132.0	118.0	112.0	N
1949.666667	136	148.0	148.0	135.0	121.0	129.0	132.0	118.0	1
1949.750000	119	136.0	148.0	148.0	135.0	121.0	129.0	132.0	1

10 rows × 21 columns

In [31]:

```
# compute autocovariance for a single lag
air.AirPassengers.cov(air.lag1)
```

Out[31]:

```
13741.562198365013
```

In [32]:

```
# compute autocovariance for multiple lags
cov_list = []
lags = range(1,21)
for i in lags:
    covariance = air.AirPassengers.cov(air[f"lag{i}"])
    cov_list.append(covariance)
print(cov_list)
```

```
[13741.562198365013, 12784.257167116171, 11869.19295845
9978, 11165.11798561151, 10685.684130956106, 10307.2694
38273564, 10122.690103048517, 10111.148583877997, 1045
0.690768380318, 11051.489339019188, 11800.03964456596,
12196.593627110802, 11574.180152671757, 10641.442218246
868, 9794.949794089149, 9149.189468503937, 8674.3088988
8764, 8309.871999999996, 8128.054451612902, 8114.742460
005246]
```

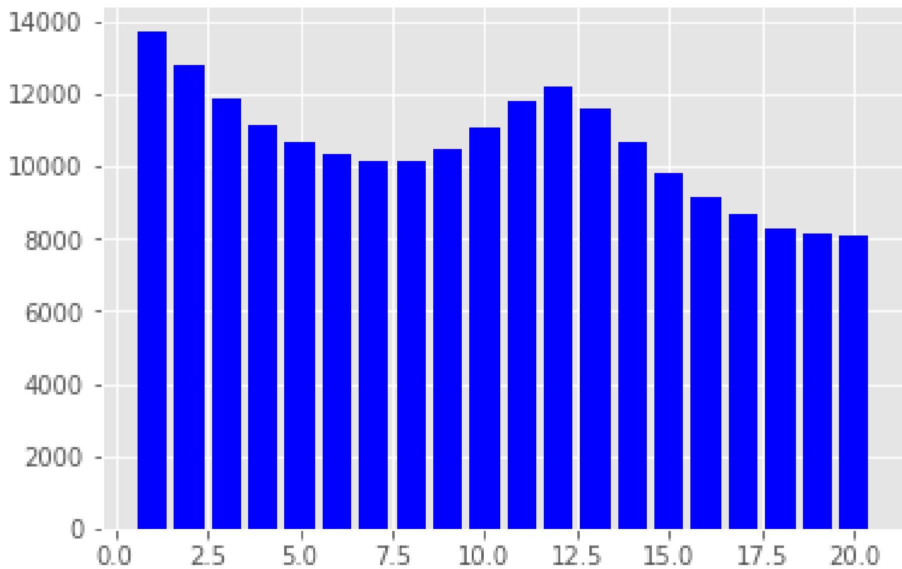
In [33]:

```
corr_list = []
lags = range(1,21)
for i in lags:
    correlation = air.AirPassengers.corr(air[f"lag{i}"])
    corr_list.append(correlation)
print(corr_list)
```

```
[0.9601946480498522, 0.8956753113926392, 0.837394765081
794, 0.7977346989350624, 0.7859431491184304, 0.78391879
59206183, 0.7845921291388301, 0.7922150472595746, 0.827
8519011167602, 0.8827127951607842, 0.9497020331006317,
0.9905273692085446, 0.9481066160592017, 0.8754477915539
791, 0.8114659384543108, 0.7694487920842656, 0.75581912
30371455, 0.7487523142605247, 0.7455000168641182, 0.751
7886585378154]
```

In [34]:

```
# autocovariance function (correlogram)
plt.bar(lags, cov_list, color="b");
```



In [35]:

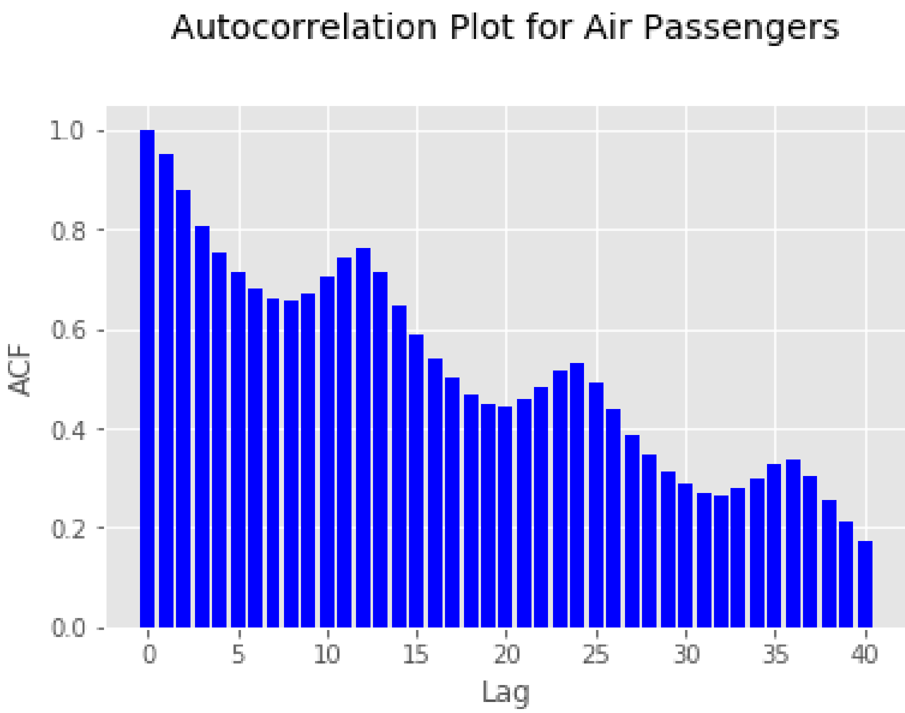
```
# create the autocorrelation function from
autocorr = acf(air.AirPassengers, nlags=40, fft=False)
autocorr
```

Out[35]:

```
array([1.        , 0.94804734, 0.87557484, 0.80668116,
0.75262542,
       0.71376997, 0.6817336 , 0.66290439, 0.65561048,
0.67094833,
       0.70271992, 0.74324019, 0.76039504, 0.71266087,
0.64634228,
       0.58592342, 0.53795519, 0.49974753, 0.46873401,
0.44987066,
       0.4416288 , 0.45722376, 0.48248203, 0.51712699,
0.53218983,
       0.49397569, 0.43772134, 0.3876029 , 0.34802503,
0.31498388,
       0.28849682, 0.27080187, 0.26429011, 0.27679934,
0.2985215 ,
       0.32558712, 0.3370236 , 0.30333486, 0.25397708,
0.21065534,
       0.17217092])
```

In [36]:

```
plt.bar(range(len(autocorr)), autocorr, color="b")
plt.title("Autocorrelation Plot for Air Passengers", y=1.1)
plt.xlabel("Lag")
plt.ylabel("ACF");
```



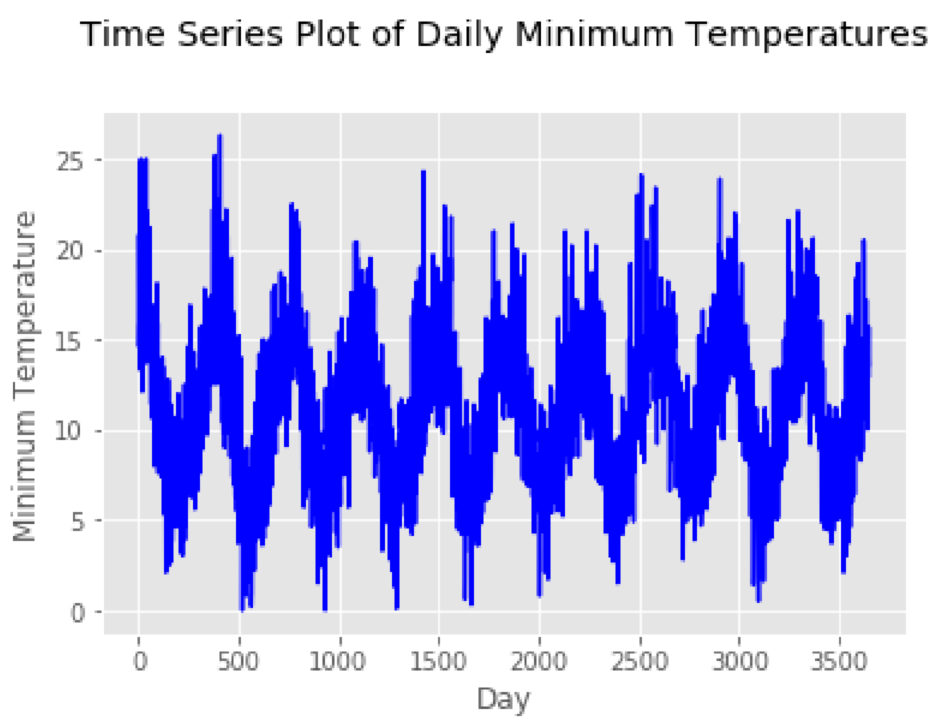
In [37]:

```
# we can also decompose() method in statsmodel
# to check for seasonality and trend
#statsmodels.tsa.seasonal.seasonal_decompose

from statsmodels.tsa.seasonal import seasonal_decompose
decomp = seasonal_decompose(air_passenger["AirPassengers"],
                            model='additive', freq=30)
decomp.plot();
```

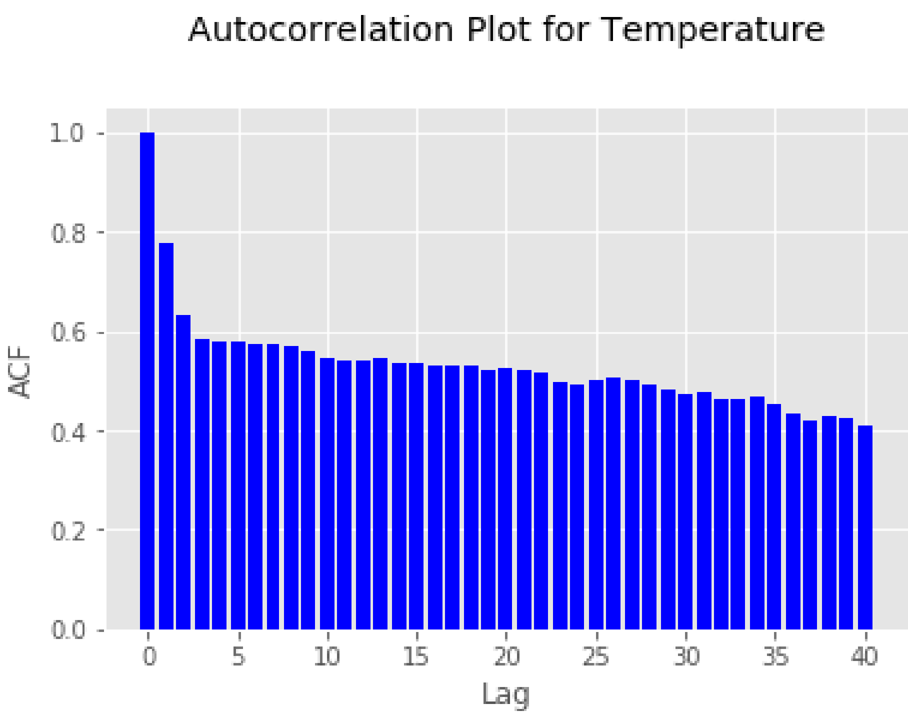
In [38]:

```
url = r"https://raw.githubusercontent.com/jbrownlee/Datasets/master/
temperature = pd.read_csv(url)
temperature.set_index("Date")
plt.title("Time Series Plot of Daily Minimum Temperatures", y=1.1)
plt.xlabel("Day")
plt.ylabel("Minimum Temperature")
plt.plot(temperature.index, temperature.Temp, color="b");
```



In [39]:

```
acf_temp = acf(temperature.Temp,  nlags=40, fft=False)
plt.bar(range(len(acf_temp)), acf_temp, color="b")
plt.title("Autocorrelation Plot for Temperature", y=1.1)
plt.xlabel("Lag")
plt.ylabel("ACF");
```



In [40]:

```
# AR(3): AR model of order 3
temp_AR = temperature.copy()
for i in range(1, 4):
    temp_AR[f"temp_{i}"] = temp_AR["Temp"].shift(i)
temp_AR.head(10)
```

Out[40]:

	Date	Temp	temp_1	temp_2	temp_3
0	1981-01-01	20.7	NaN	NaN	NaN
1	1981-01-02	17.9	20.7	NaN	NaN
2	1981-01-03	18.8	17.9	20.7	NaN
3	1981-01-04	14.6	18.8	17.9	20.7
4	1981-01-05	15.8	14.6	18.8	17.9
5	1981-01-06	15.8	15.8	14.6	18.8
6	1981-01-07	15.8	15.8	15.8	14.6
7	1981-01-08	17.4	15.8	15.8	15.8
8	1981-01-09	21.8	17.4	15.8	15.8
9	1981-01-10	20.0	21.8	17.4	15.8

In [41]:

```
from statsmodels.formula.api import ols

formula = "Temp ~ temp_1 + temp_2 + temp_3"
model = ols(formula, data=temp_AR).fit()
model.summary2()
```

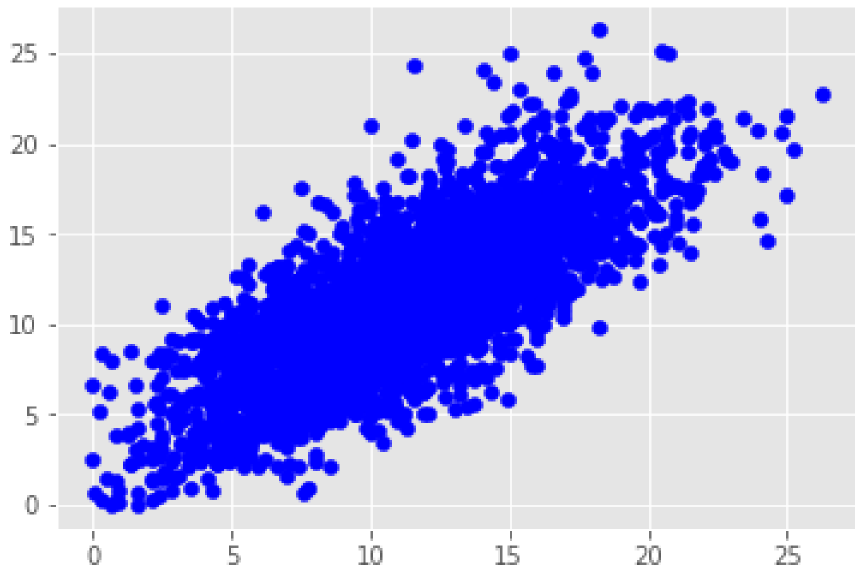
Date:	2020-02-10 09:32	BIC:	17117.9514
No. Observations:	3647	Log-Likelihood:	-8542.6
Df Model:	3	F-statistic:	1953.
Df Residuals:	3643	Prob (F-statistic):	0.00
R-squared:	0.617	Scale:	6.3467

	Coef.	**Std.Err.**	**t**	**P>|t|**	**[0.025**	**0.975]**
Intercept	1.8882	0.1340	14.0938	0.0000	1.6255	2.1508
temp_1	0.7000	0.0163	43.0404	0.0000	0.6681	0.7319
temp_2	-0.0594	0.0200	-2.9766	0.0029	-0.0985	-0.0203
temp_3	0.1902	0.0163	11.6995	0.0000	0.1583	0.2221

Omnibus:	8.194	Durbin-Watson:	2.056
Prob(Omnibus):	0.017	Jarque-Bera (JB):	9.630

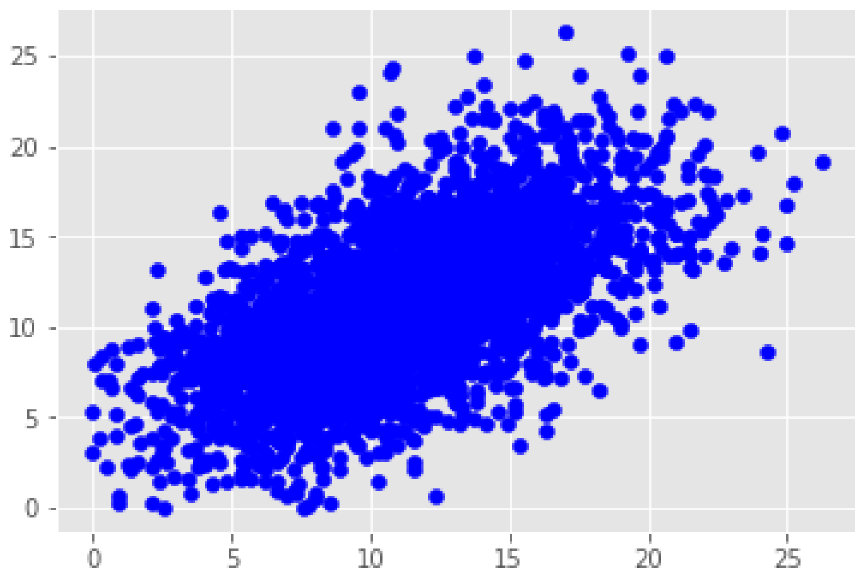
In [42]:

```
# check autocorrelation
plt.scatter(temp_AR.Temp, temp_AR.temp_1, c="b");
```



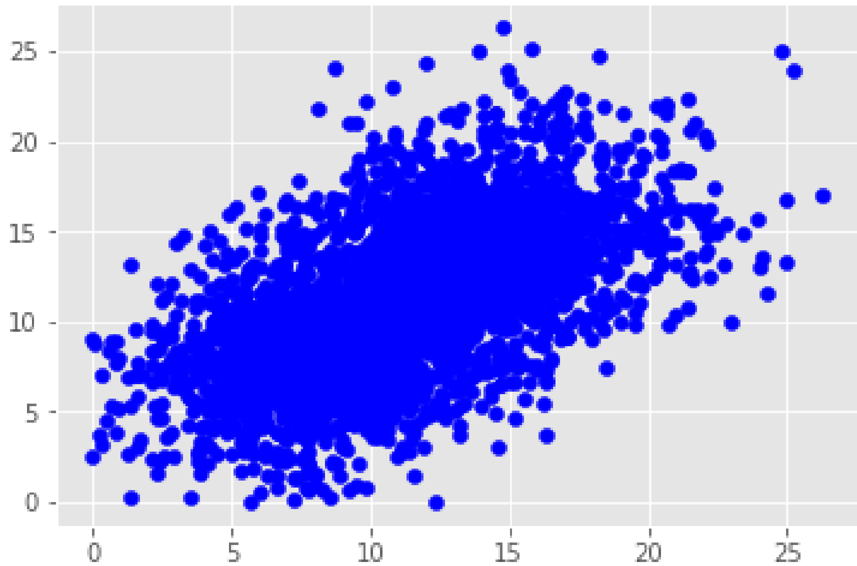
In [43]:

```
plt.scatter(temp_AR.Temp, temp_AR.temp_2, c="b");
```



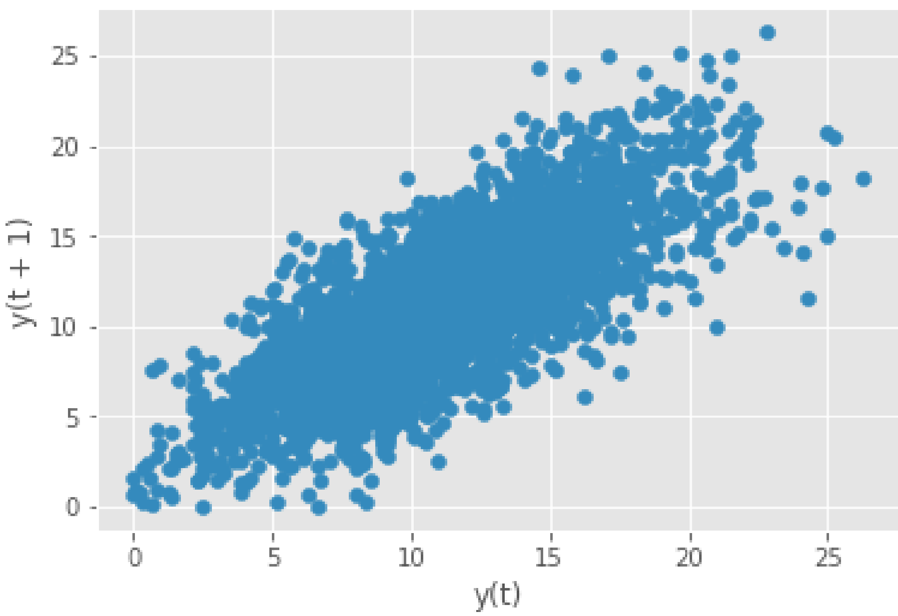
In [44]:

```
plt.scatter(temp_AR.Temp, temp_AR.temp_3, c="b");
```



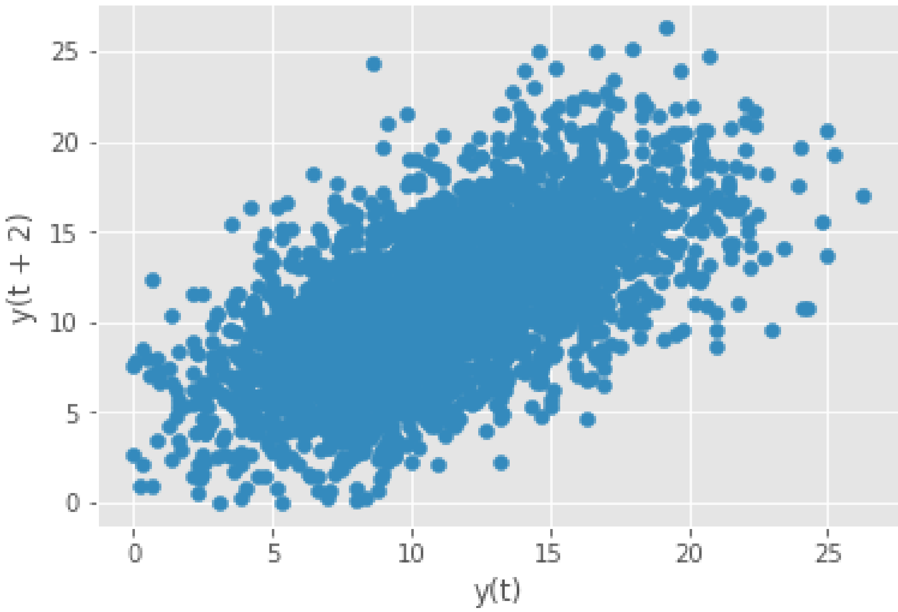
In [45]:

```
# visualize correlation between data and lag variables
# using pandas plotting features
pd.plotting.lag_plot(temp_AR.Temp, lag=1);
```



In [46]:

```
pd.plotting.lag_plot(temp_AR.Temp, lag=2);
```



In [47]:

```
# check correlation among time series and lags
temp_AR.corr()
```

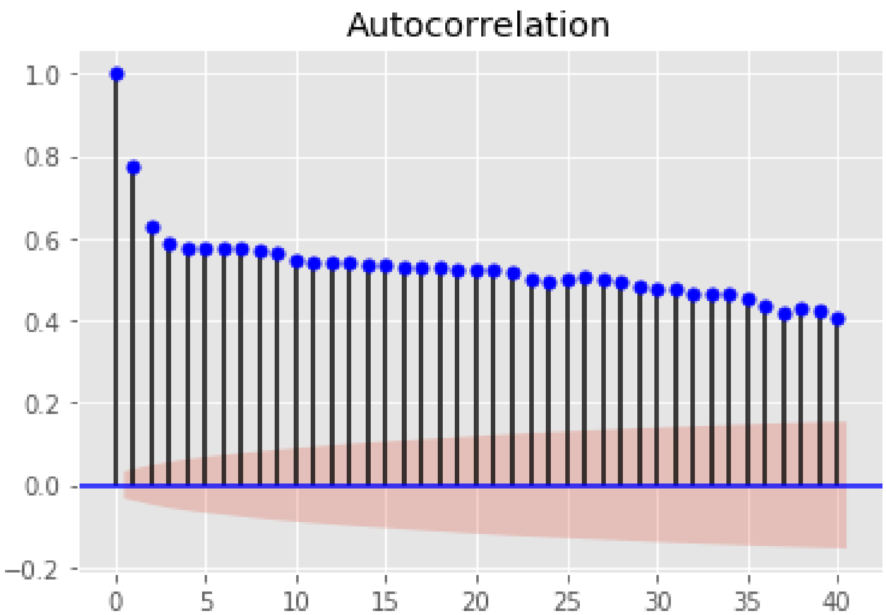
Out[47]:

	Temp	temp_1	temp_2	temp_3
Temp	1.000000	0.774870	0.631119	0.586375
temp_1	0.774870	1.000000	0.774886	0.631095
temp_2	0.631119	0.774886	1.000000	0.774878
temp_3	0.586375	0.631095	0.774878	1.000000

In [48]:

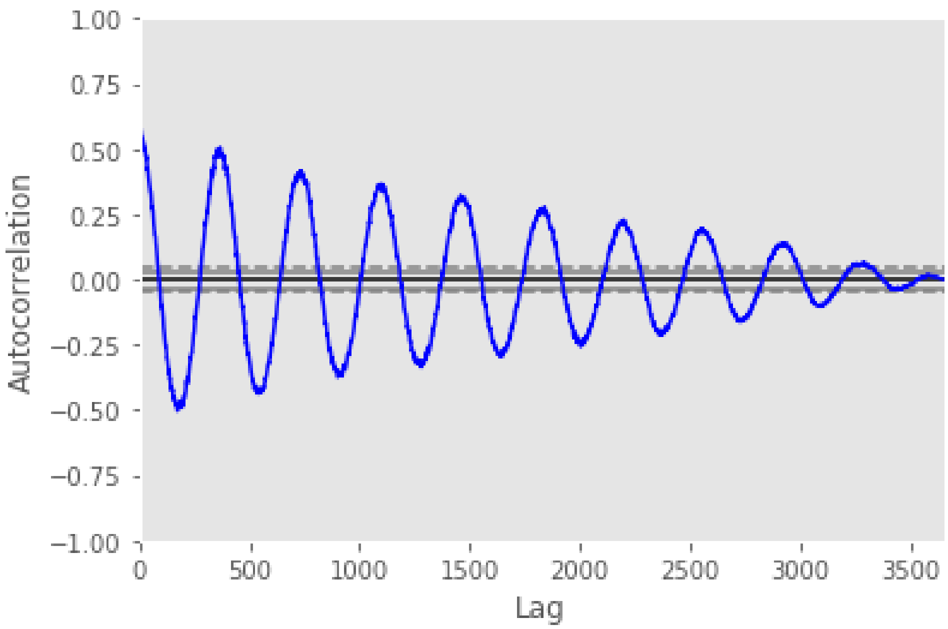
```python
# plot an autocorrelation function using statsmodels

plot_acf(temp_AR.Temp, lags=40, c="b");
```



In [49]:

```
# plot an autocorrelation function using pandas
from pandas.plotting import autocorrelation_plot
autocorrelation_plot(temp_AR.Temp, c="b");
```

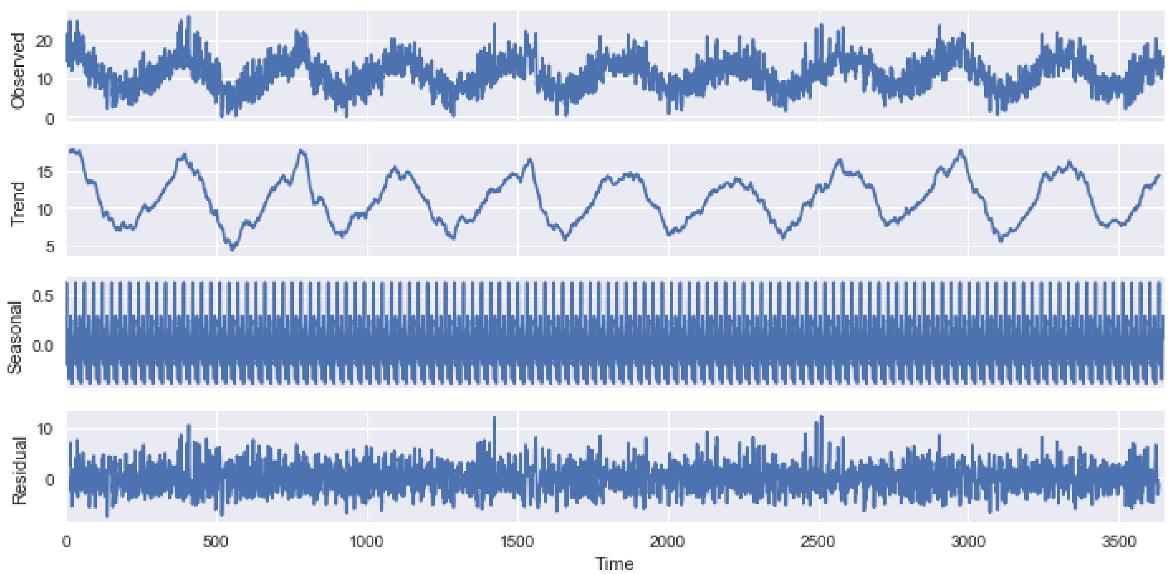


In [50]:

```
plt.style.use("seaborn")
```

In [51]:

```
# use statsmodels.tsa.seasonal.seasonal_decompose
# to decompose visual plots
plt.rcParams['figure.figsize']=(10,5)
decomp = seasonal_decompose(temp_AR["Temp"].values,
                            model='additive', freq=30)
decomp.plot();
```



In [52]:

```
# perform the Dickey-Fuller test to statistically test for stationar
from statsmodels.tsa.stattools import adfuller

test = adfuller(temp_AR.Temp, autolag="AIC")
pd.DataFrame(test[0:4], index=["teststat", "p-value",
                               "#lags used", "#observations used"])
```

Out[52]:

	0
teststat	-4.444805
p-value	0.000247
#lags used	20.000000
#observations used	3629.000000

In [ ]:

```

```

In [53]:

```
df = pd.DataFrame({"A":[1, 3, 10, 15, 5, 20]})
df["diff"] = df.diff(periods=1)
df
```

Out[53]:

	A	diff
0	1	NaN
1	3	2.0
2	10	7.0
3	15	5.0
4	5	-10.0
5	20	15.0

In [54]:

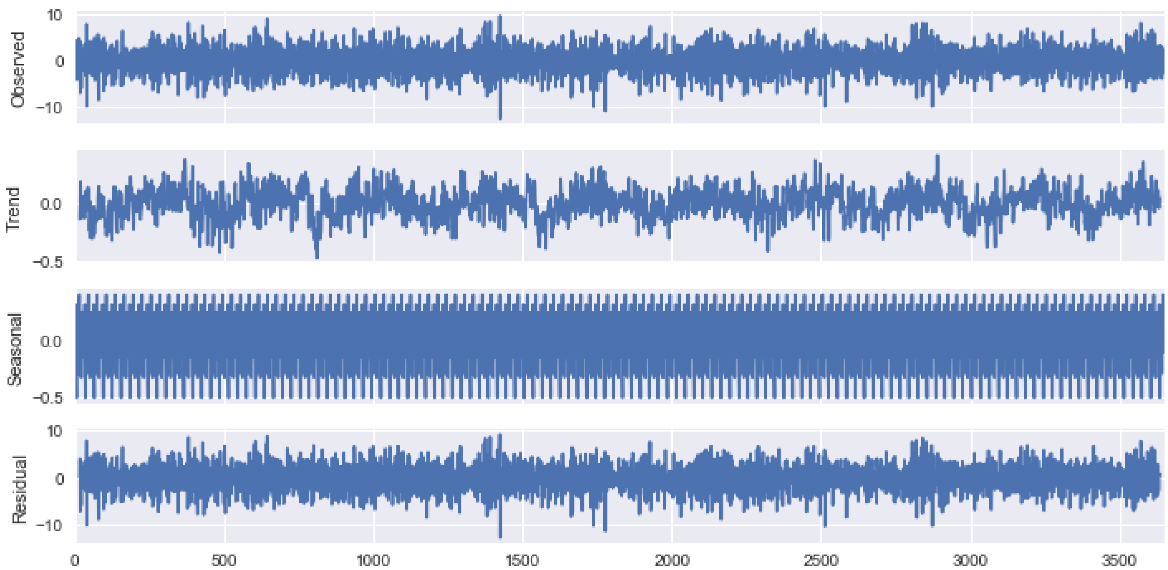
```
# use pandas diff() method to find the
# difference between observations
temp_AR["stationary"] = temp_AR["Temp"].diff()
temp_AR[["Temp", "stationary"]].head()
```

Out[54]:

	Temp	stationary
0	20.7	NaN
1	17.9	-2.8
2	18.8	0.9
3	14.6	-4.2
4	15.8	1.2

In [55]:

```
# check stationarity again with the transformed data
decomp = seasonal_decompose(temp_AR["stationary"].dropna(),
                            model='additive', freq=30)
decomp.plot();
```



In [ ]:

```

```

In [ ]:

```

```

Fit the AR Model

In [56]:

```
# split data into train and test set
X = temp_AR['Temp'].dropna()
X = X.values # extract only values without indexes
n = len(X)
n
```

Out[56]:

```
3650
```

In [57]:

```
# prediction will be made for the last 7 days
train_data = X[1:n-7]
test_data = X[n-7:]
```

In [58]:

```
#train the autoregression model
model = AR(train_data).fit()
```

In [59]:

```
# lag value selected
model.k_ar
```

Out[59]:

```
29
```

In [60]:

```
# model parameters
model.params
```

Out[60]:

```
array([ 5.57543506e-01,  5.88595221e-01, -9.08257090e-0
2,  4.82615092e-02,
        4.00650265e-02,  3.93020055e-02,  2.59463738e-0
2,  4.46675960e-02,
        1.27681498e-02,  3.74362239e-02, -8.11700276e-0
4,  4.79081949e-03,
        1.84731397e-02,  2.68908418e-02,  5.75906178e-0
4,  2.48096415e-02,
        7.40316579e-03,  9.91622149e-03,  3.41599123e-0
2, -9.11961877e-03,
        2.42127561e-02,  1.87870751e-02,  1.21841870e-0
2, -1.85534575e-02,
       -1.77162867e-03,  1.67319894e-02,  1.97615668e-0
2,  9.83245087e-03,
        6.22710723e-03, -1.37732255e-03])
```

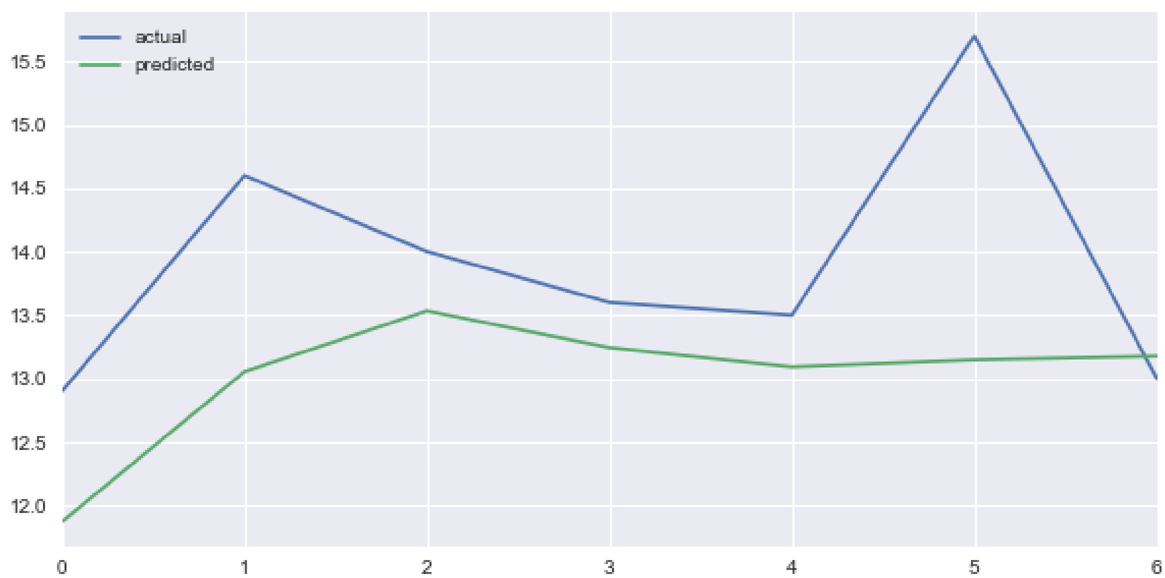
In [61]:

```
# make predictions

predictions = model.predict(start=len(train_data),
                            end=len(train_data) + len(test_data)-1,
                            dynamic=False)

test_pred = pd.concat([pd.DataFrame(test_data),
                       pd.DataFrame(predictions)], axis="columns")

test_pred.columns = ["actual", "predicted"]
test_pred.plot();
```



In [62]:

```
# overall prediction error
from sklearn.metrics import mean_squared_error
mean_squared_error(test_data, predictions)
```

Out[62]:

```
1.5015252310069296
```

In [63]:

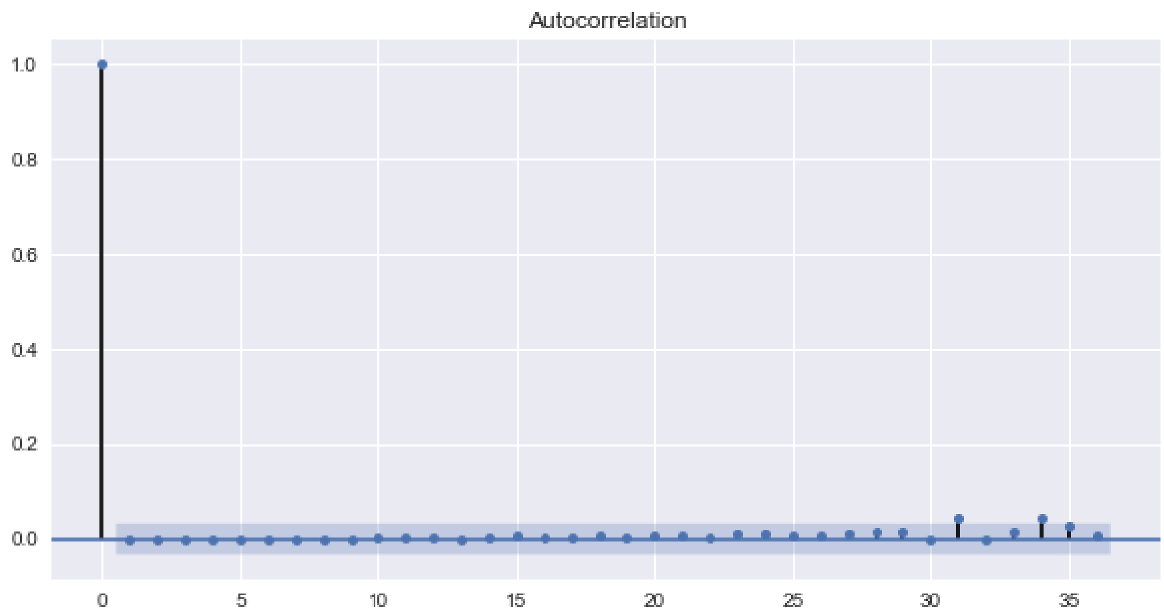
```
# check if residuals are normally distributed
residuals = model.resid
residuals
```

Out[63]:

```
array([-0.68105821, -0.69572786,  2.74223153, ..., -0.0
726287 ,
        0.15670338, -4.17767517])
```

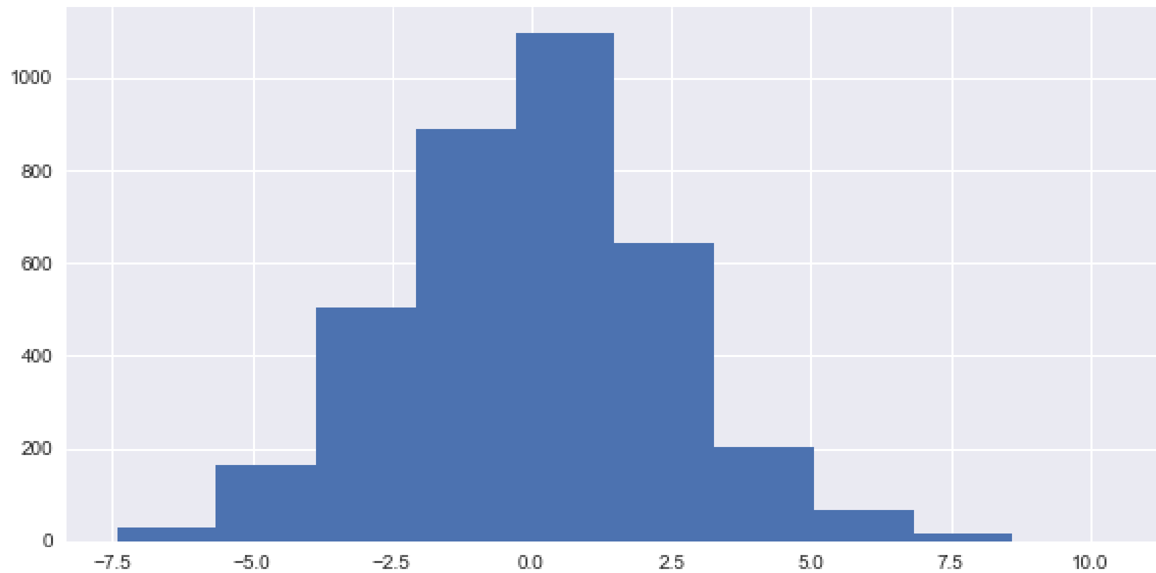
In [64]:

```
plot_acf(residuals);
```



In [65]:

```
plt.hist(residuals);
```



In [66]:

```
np.mean(residuals)
```

Out[66]:

```
8.661030191772713e-15
```

Fitting Moving Average (MA) Model

In [67]:

```
# split data into train and test set
X = temp_AR['Temp'].dropna()
X = X.values # extract only values without indexes
n = len(X)

# prediction will be made for the last 7 days
train_data = X[1:n-7]
test_data = X[n-7:]
```

In [68]:

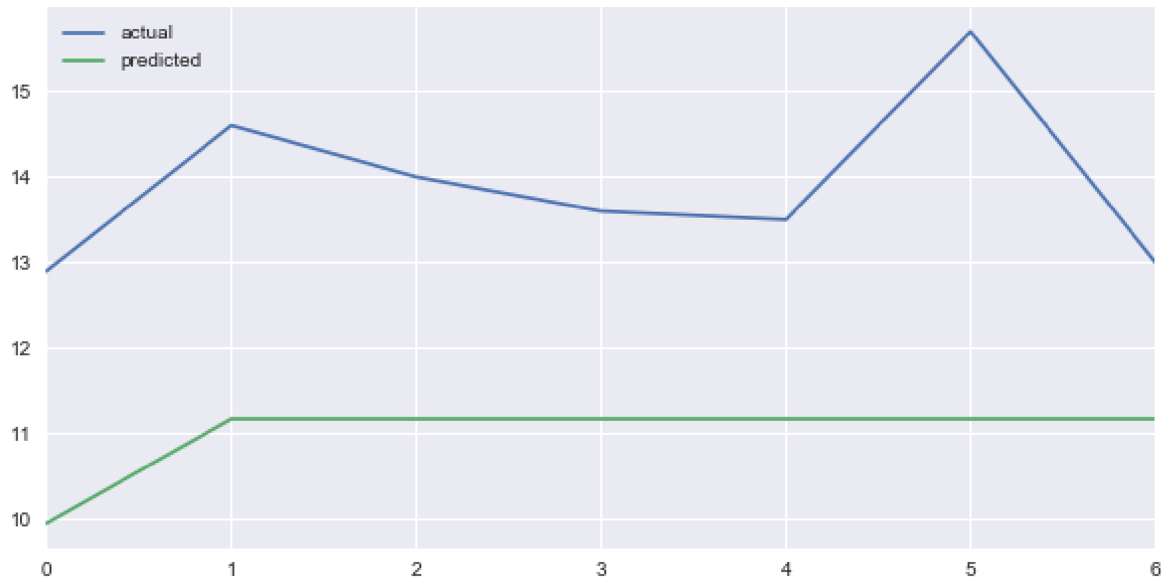
```
# fit MA model
model_MA = ARMA(train_data, order=(0,1)).fit(disp=False)

# make predictions
predict_MA = model_MA.predict(start=len(train_data),
                              end=len(train_data) + len(test_data)-1,
                              dynamic=False)
```

In [69]:

```
test_pred_MA = pd.concat([pd.DataFrame(test_data),
                          pd.DataFrame(predict_MA)], axis="columns")

test_pred_MA.columns = ["actual", "predicted"]
test_pred_MA.plot();
```



In [70]:

```
# MSE
mean_squared_error(test_data, predict_MA)
```

Out[70]:

9.096985026186768

In [71]:

```
# AIC
model_MA.aic
```

Out[71]:

```
18511.488006977022
```

In [72]:

```
# use all data for training
# make prediction for the next period
mod = ARMA(X, order=(0,1)).fit(disp=False)
mod.predict(start=len(X), end=len(X), dynamic=False)
```

Out[72]:

```
array([10.91151172])
```

Fitting ARMA Model

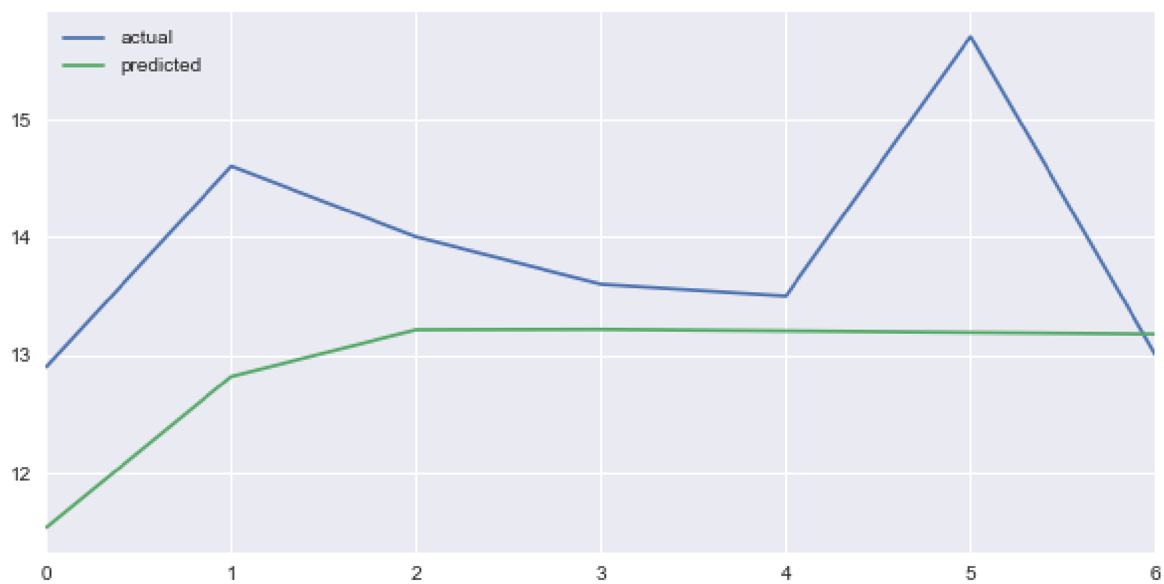
In [73]:

```
# fit ARMA model
model_ARMA = ARMA(train_data, order=(2,3)).fit(disp=False)

# make predictions
predict_ARMA = model_ARMA.predict(start=len(train_data),
                                  end=len(train_data) + len(test_data)-1,
                                  dynamic=False)

test_pred_ARMA = pd.concat([pd.DataFrame(test_data),
                           pd.DataFrame(predict_ARMA)], axis="columns")

test_pred_ARMA.columns = ["actual", "predicted"]
test_pred_ARMA.plot();
```



In [74]:

```
# MSE
mean_squared_error(test_data, predict_ARMA)
```

Out[74]:

1.7470674403217132

In [75]:

```
# AIC
model_ARMA.aic
```

Out[75]:

16748.03006032794

Fit an ARIMA Model

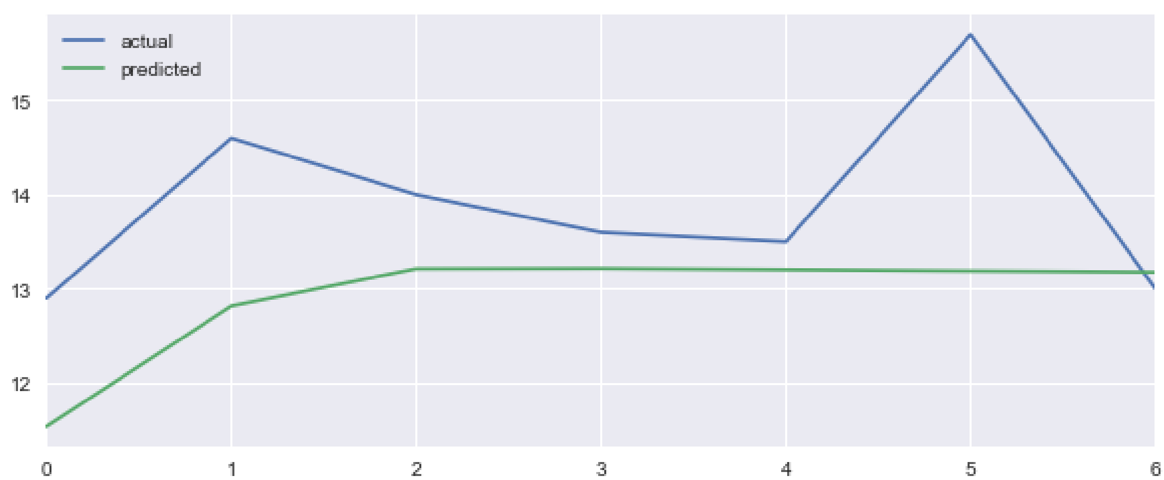
In [76]:

```
# fit ARIMA model
model_ARIMA = ARIMA(train_data, order=(2, 0, 3)).fit(disp=False)

# make predictions
predict_ARIMA = model_ARIMA.predict(start=len(train_data),
                                    end=len(train_data) + len(test_data)-1,
                                    dynamic=False)

test_pred_ARIMA = pd.concat([pd.DataFrame(test_data),
                             pd.DataFrame(predict_ARIMA)], axis="columns")

test_pred_ARIMA.columns = ["actual", "predicted"]
test_pred_ARIMA.plot(figsize=(10, 4));
```



In [77]:

```
# MSE
mean_squared_error(test_data, predict_ARMA)
```

Out[77]:

1.7470674403217132

In [78]:

```
# AIC
model_ARIMA.aic
```

Out[78]:

16748.03006032794

Fit an ARIMA Model on Non-stationary Data

In [79]:

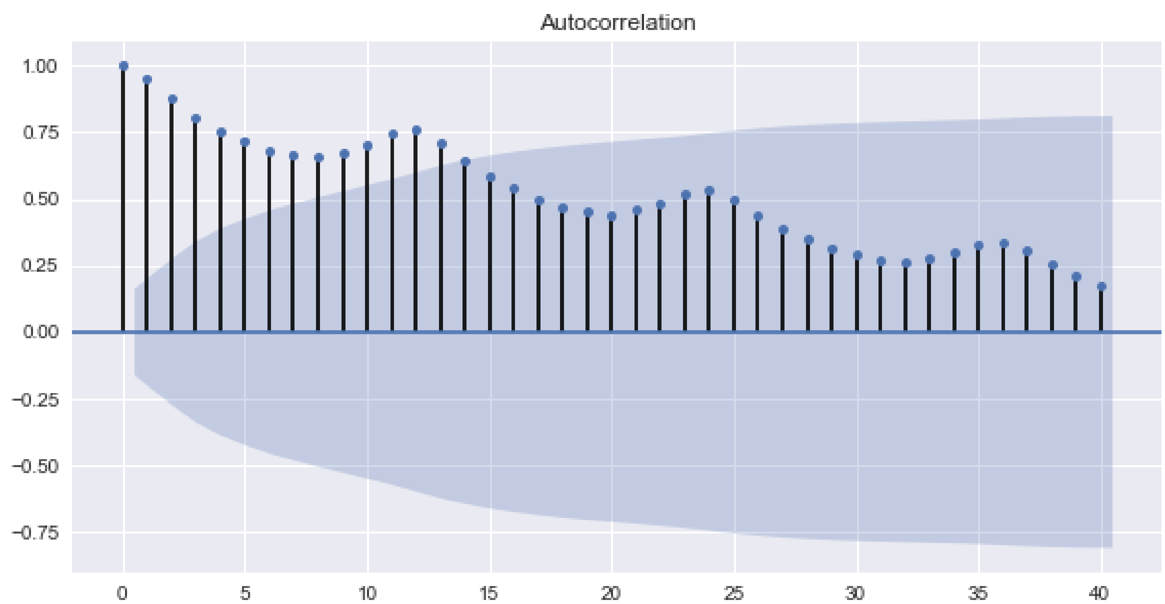
```
air_pass.head()
```

Out[79]:

	AirPassengers
time	
1949.000000	112
1949.083333	118
1949.166667	132
1949.250000	129
1949.333333	121

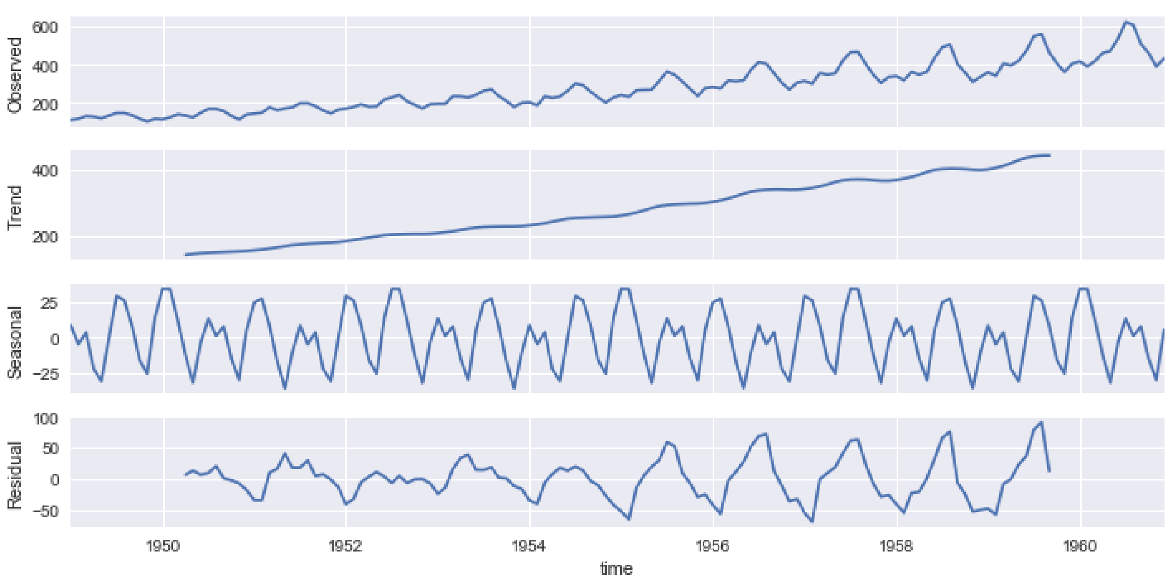
In [80]:

```
# Autocorrelation Function
plot_acf(air_pass.AirPassengers, lags=40);
```



In [81]:

```
decomp = seasonal_decompose(air_pass["AirPassengers"],
                            model='additive', freq=30)
decomp.plot();
```



In [82]:

```
# split data into train and test set
X_air = air_pass['AirPassengers'].dropna()
X_air = X_air.values # extract only values without indexes
n = len(X_air)

# prediction will be made for the last 11 days
# since a new season seems to begins every 12th day
train = X_air[1:n-11]
test = X_air[n-11:]
```

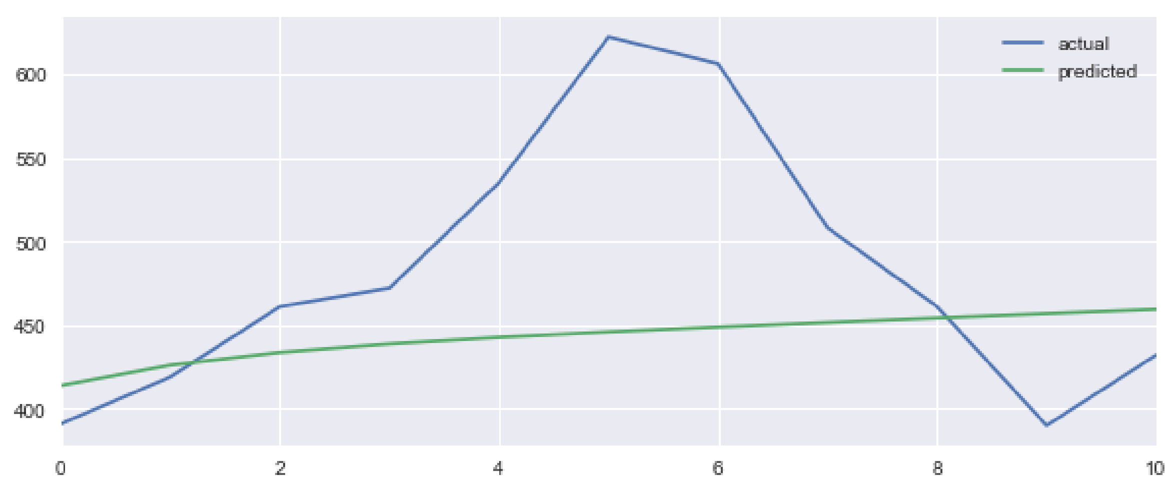
In [83]:

```
# fit ARIMA model
model_ARIMA2 = ARIMA(train, order=(1, 1, 2)).fit(disp=False)

# make predictions
predict_ARIMA2 = model_ARIMA2.predict(start=len(train),
                            end=len(train) + len(test)-1,
                            dynamic=False, typ="levels")

test_pred_ARIMA2 = pd.concat([pd.DataFrame(test),
                       pd.DataFrame(predict_ARIMA2)], axis="columns"

test_pred_ARIMA2.columns = ["actual", "predicted"]
test_pred_ARIMA2.plot(figsize=(10, 4));
```



In [84]:

```
# MSE
mean_squared_error(test, predict_ARIMA2)
```

Out[84]:

6830.152050985579

In [85]:

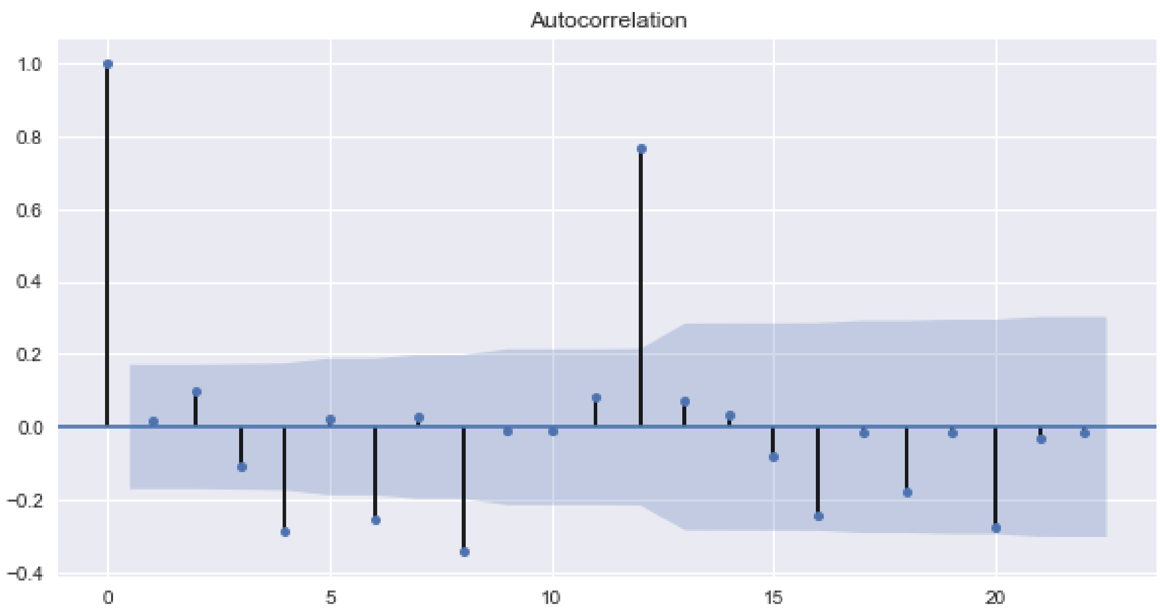
```
# AIC
model_ARIMA2.aic
```

Out[85]:

1240.732331226167

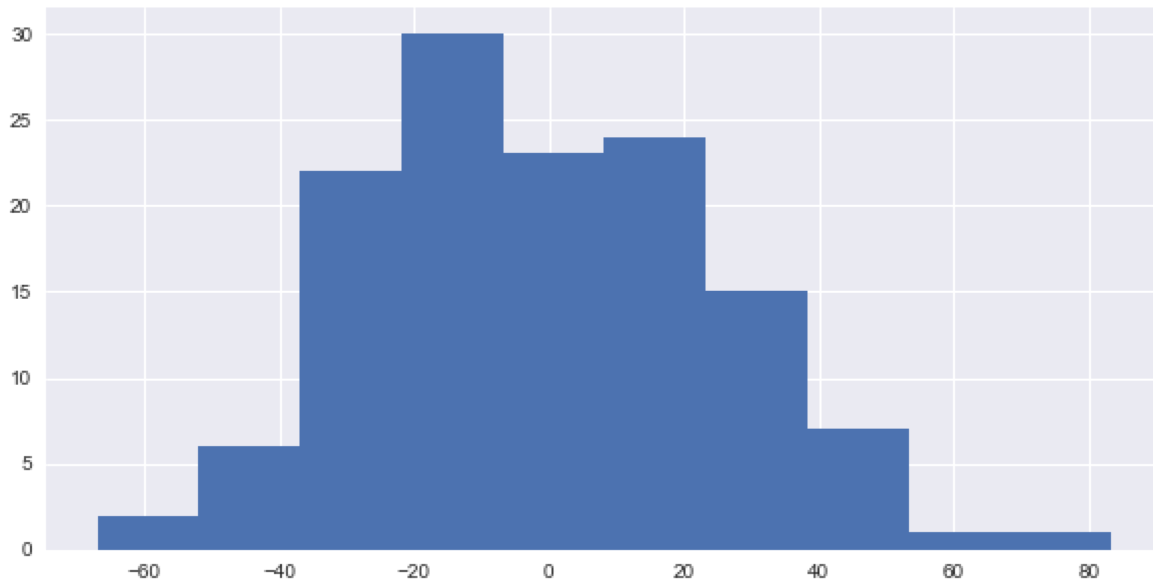
In [86]:

```
# residual ACF
plot_acf(model_ARIMA2.resid);
```



In [87]:

```
plt.hist(model_ARIMA2.resid);
```



In [88]:

```
# split data into train and test set
X_air= air_pass['AirPassengers'].dropna()
X_air =X_air.values # extract only values without indexes
n = len(X_air)

# prediction will be made for the last 11 days
# since a new season seems to begins every 12th day
train_log = np.log(X_air[1:n-11])
test_log = np.log(X_air[n-11:])
```

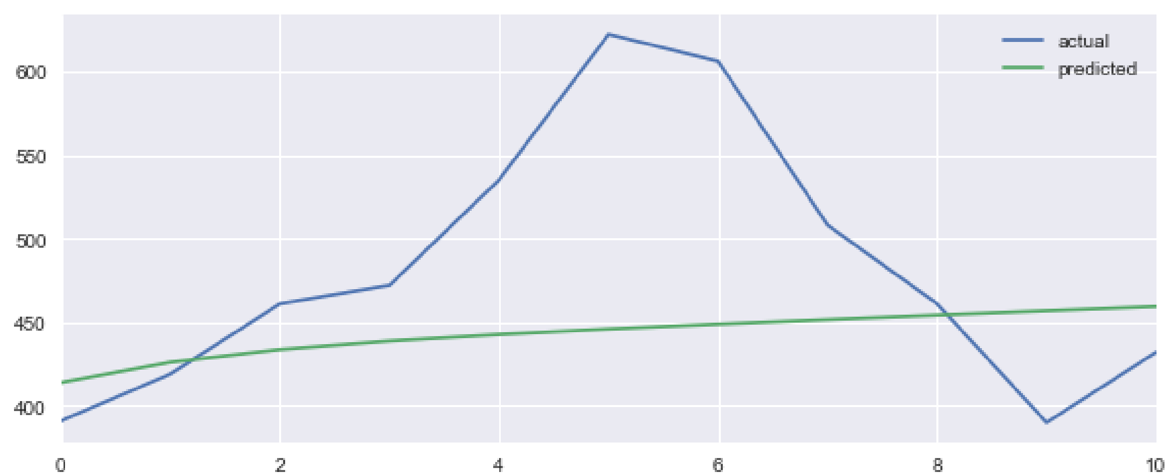
In [89]:

```
# fit ARIMA model
model_ARIMA3 = ARIMA(train, order=(1, 1, 2)).fit(disp=False)

# make predictions
predict_ARIMA3 = model_ARIMA3.predict(start=len(train_log),
                            end=len(train_log) + len(test_log)-1,
                            dynamic=False, typ="levels")
# specify typ="levels" to keep predictions to same scale as original
# data in a case where differencing was performed.

test_pred_ARIMA3 = pd.concat([pd.DataFrame(X_air[n-11:]),# use origi
                       pd.DataFrame(predict_ARIMA3)], axis="columns"

test_pred_ARIMA3.columns = ["actual", "predicted"]
test_pred_ARIMA3.plot(figsize=(10, 4));
```



In [90]:

```
# MSE
mean_squared_error(test_log, predict_ARIMA3)
```

Out[90]:

190872.11608371304

In [91]:

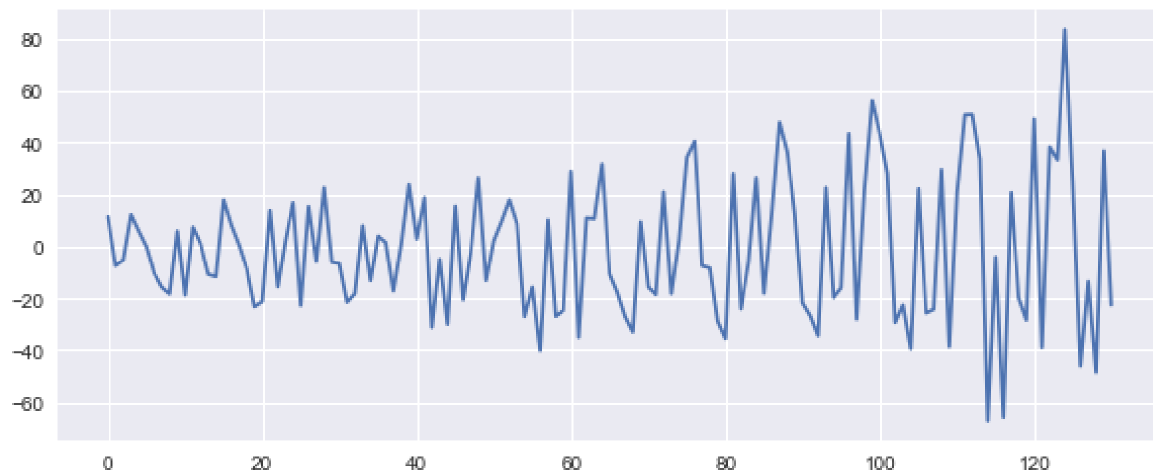
```
model_ARIMA3.aic
```

Out[91]:

1240.732331226167

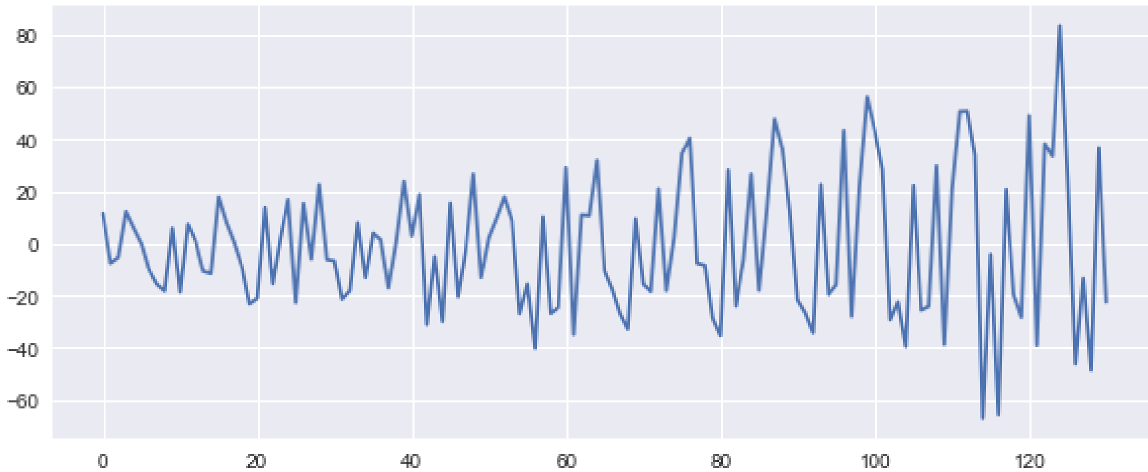
In [92]:

```
# line plot of residuals
plt.figure(figsize=(10, 4))
plt.plot(model_ARIMA3.resid);
```



In [93]:

```
# line plot of residuals
plt.figure(figsize=(10, 4))
plt.plot(model_ARIMA2.resid);
```



Results of Model Fit and Estimate

In [94]:

```
model_ARIMA2.summary()
```

Out[94]:

ARIMA Model Results

Dep. Variable:	D.y	**No. Observations:**	131
Model:	ARIMA(1, 1, 2)	**Log Likelihood**	-615.366
Method:	css-mle	**S.D. of innovations**	26.160
Date:	Mon, 10 Feb 2020	**AIC**	1240.732
Time:	09:33:00	**BIC**	1255.108
Sample:	1	**HQIC**	1246.574

	coef	std err	z	P>|z|	[0.025	0.975]
const	2.5425	0.188	13.519	0.000	2.174	2.911
ar.L1.D.y	0.5205	0.103	5.039	0.000	0.318	0.723
ma.L1.D.y	-0.4416	0.127	-3.479	0.001	-0.690	-0.193
ma.L2.D.y	-0.5583	0.126	-4.442	0.000	-0.805	-0.312

Roots

	Real	Imaginary	Modulus	Frequency
AR.1	1.9211	+0.0000j	1.9211	0.0000
MA.1	1.0000	+0.0000j	1.0000	0.0000
MA.2	-1.7911	+0.0000j	1.7911	0.5000

In [95]:

```
model_ARIMA3.summary()
```

Out[95]:

ARIMA Model Results

Dep. Variable:	D.y	**No. Observations:**	131
Model:	ARIMA(1, 1, 2)	**Log Likelihood**	-615.366
Method:	css-mle	**S.D. of innovations**	26.160
Date:	Mon, 10 Feb 2020	**AIC**	1240.732
Time:	09:33:01	**BIC**	1255.108
Sample:	1	**HQIC**	1246.574

	coef	std err	z	P>|z|	[0.025	0.975]
const	2.5425	0.188	13.519	0.000	2.174	2.911
ar.L1.D.y	0.5205	0.103	5.039	0.000	0.318	0.723
ma.L1.D.y	-0.4416	0.127	-3.479	0.001	-0.690	-0.193
ma.L2.D.y	-0.5583	0.126	-4.442	0.000	-0.805	-0.312

Roots

	Real	Imaginary	Modulus	Frequency
AR.1	1.9211	+0.0000j	1.9211	0.0000
MA.1	1.0000	+0.0000j	1.0000	0.0000
MA.2	-1.7911	+0.0000j	1.7911	0.5000

In [96]:

```
test_pred_ARIMA3.tail()
```

Out[96]:

	actual	predicted
6	606	448.795751
7	508	451.529410
8	461	454.171393
9	390	456.765654
10	432	459.335075

In [97]:

```
len(model_ARIMA3.predict())
```

Out[97]:

131

In [ ]:

```
```

In [98]:

```
import pandas_datareader as pdr

plt.figure(figsize=(10, 4))
google = pdr.get_data_yahoo("AAPL", "2019-01-30")
plt.plot(google.index, google.Close);
```

In [99]:

```
# another way to get data from the web using the data.Datareader in
pdr.data.DataReader("AMZN", data_source="yahoo", start="2019-01-01",
```

Out[99]:

Date	High	Low	Open	Close	Volume	Adj Close
2019-01-02	1553.359985	1460.930054	1465.199951	1539.130005	7983100	1539.1300
2019-01-03	1538.000000	1497.109985	1520.010010	1500.280029	6975600	1500.2800
2019-01-04	1594.000000	1518.310059	1530.000000	1575.390015	9182600	1575.3900
2019-01-07	1634.560059	1589.189941	1602.310059	1629.510010	7993200	1629.5100
2019-01-08	1676.609985	1616.609985	1664.689941	1656.579956	8881400	1656.5799

In [100]:

```
# pull data for more than one company
pdr.get_data_yahoo(["AMZN", "GOOG"], "2008-02-01", "2009-02-01").hea
```

Out[100]:

Attributes	Adj Close		Close		High	
Symbols	AMZN	GOOG	AMZN	GOOG	AMZN	GOOG
Date						
2008-02-01	74.629997	256.986755	74.629997	256.986755	79.400002	267.332977
2008-02-04	73.949997	246.789978	73.949997	246.789978	76.660004	255.432571
2008-02-05	72.089996	252.453735	72.089996	252.453735	74.209999	253.549637
2008-02-06	68.489998	249.918243	68.489998	249.918243	72.430000	254.630585
2008-02-07	70.910004	251.532196	70.910004	251.532196	72.709999	256.134949

In [101]:

```
# another way of pulling stock data from the internet using
# the data.DataReader() in pandas_datareader
pdr.data.DataReader("AMZN", data_source="yahoo",
                    start="2019-01-01", end="2020-01-01").head()
```

Out[101]:

Date	High	Low	Open	Close	Volume	Adj Close
2019-01-02	1553.359985	1460.930054	1465.199951	1539.130005	7983100	1539.1300
2019-01-03	1538.000000	1497.109985	1520.010010	1500.280029	6975600	1500.2800
2019-01-04	1594.000000	1518.310059	1530.000000	1575.390015	9182600	1575.3900
2019-01-07	1634.560059	1589.189941	1602.310059	1629.510010	7993200	1629.5100
2019-01-08	1676.609985	1616.609985	1664.689941	1656.579956	8881400	1656.5799

In [ ]: