Using the VAR to predict and forecast USA Personal Spendings with M2 Money Stock which is a measure of USA Personal assets(Savings).The AR Model is also been used to compare its predictive power with the VAR.What happens at the end?

datasets in billions of dollard,monthly and seasonally adjusted.

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:

```
from statsmodels.tsa.stattools import adfuller
from statsmodels.tools.eval_measures import rmse,mse
from statsmodels.tsa.api import VAR
import warnings
warnings.filterwarnings('ignore')
```

In [3]:

```
dp=pd.read_csv(r'C:\Users\chumj\Downloads\PersonalSpending.csv',index_col='Date',parse_dates=True)
dp.index.freq='MS'
```

In [4]:

```
ds=pd.read_csv(r'C:\Users\chumj\Downloads\savings.csv',index_col='Date',parse_dates=True)
ds.index.freq='MS'
```

In [5]:

```
# Jointing Personal spendings with savings into a sindle Dataframe
df=dp.join(ds)
```

In [6]:

```
df.head(5)
```

Out[6]:

	Spending	Money
Date		
1995-01-01	4851.2	3492.4
1995-02-01	4850.8	3489.9
1995-03-01	4885.4	3491.1
1995-04-01	4890.2	3499.2
1995-05-01	4933.1	3524.2

In [7]:

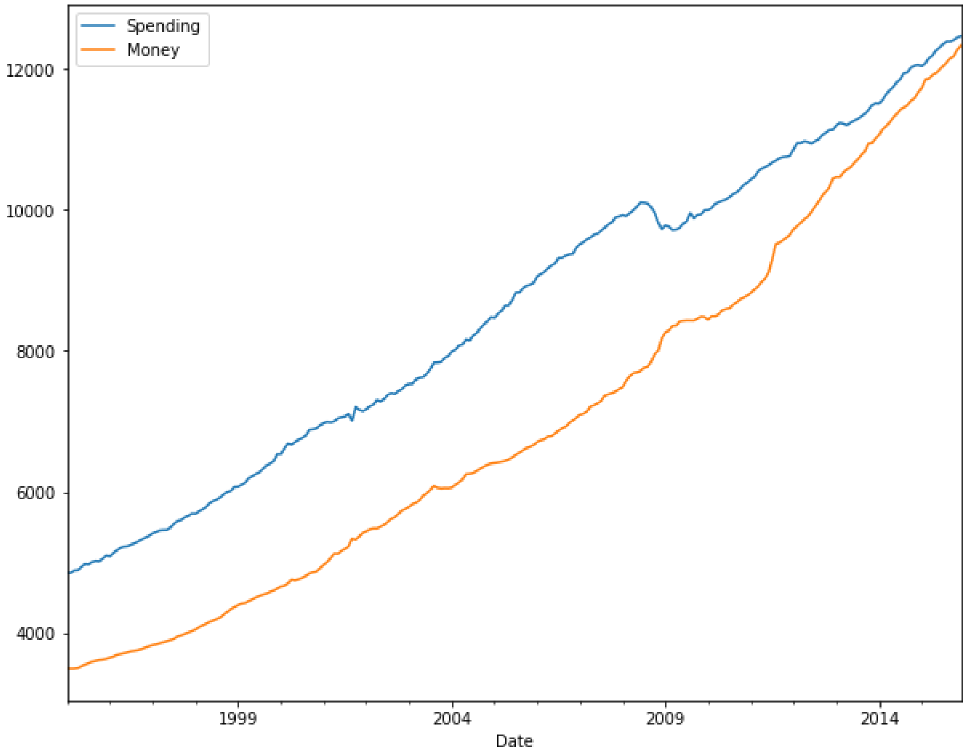
```
df=df.dropna()
df.shape
```

Out[7]:

```
(252, 2)
```

In [8]:

```
df['Spending'].plot(legend=True,figsize=(10,8))
df['Money'].plot(legend=True);
```



In [9]:

```
#Testing for stationarity using the augmented Dickey Fuller Test
from statsmodels.tsa.stattools import adfuller

def adf_test(series,title=''):
    """
    Pass in a time series and an optional title, returns an ADF report
    """
    print(f'Augmented Dickey-Fuller Test: {title}')
    result = adfuller(series.dropna(),autolag='AIC')

    labels = ['ADF test statistic','p-value','# lags used','# observations']
    out = pd.Series(result[0:4],index=labels)

    for key,val in result[4].items():
        out[f'critical value ({key})']=val

    print(out.to_string())

    if result[1] <= 0.05:
        print("Strong evidence against the null hypothesis")
        print("Reject the null hypothesis")
        print("Data has no unit root and is stationary")
    else:
        print("Weak evidence against the null hypothesis")
        print("Fail to reject the null hypothesis")
        print("Data has a unit root and is non-stationary")
```

In [10]:

```
adf_test(df['Spending'])
```

```
Augmented Dickey-Fuller Test:
ADF test statistic        0.149796
p-value                   0.969301
# lags used               3.000000
# observations          248.000000
```

```
critical value (1%)      -3.456996
critical value (5%)      -2.873266
critical value (10%)     -2.573019
Weak evidence against the null hypothesis
Fail to reject the null hypothesis
Data has a unit root and is non-stationary
```

In [11]:

```
adf_test(df['Money'])
```

```
Augmented Dickey-Fuller Test:
ADF test statistic        4.239022
p-value                   1.000000
# lags used               4.000000
# observations          247.000000
critical value (1%)      -3.457105
critical value (5%)      -2.873314
critical value (10%)     -2.573044
Weak evidence against the null hypothesis
Fail to reject the null hypothesis
Data has a unit root and is non-stationary
```

In [12]:

```
#Non of the Data is stationarity ,we take first order difference of the entire Dataframe
df_D1=df.diff().dropna()
```

In [13]:

```
adf_test(df_D1['Spending'])
```

```
Augmented Dickey-Fuller Test:
ADF test statistic     -7.226974e+00
p-value                 2.041027e-10
# lags used             2.000000e+00
# observations          2.480000e+02
critical value (1%)    -3.456996e+00
critical value (5%)    -2.873266e+00
critical value (10%)   -2.573019e+00
Strong evidence against the null hypothesis
Reject the null hypothesis
Data has no unit root and is stationary
```

In [14]:

```
adf_test(df_D1['Money'])
```

```
Augmented Dickey-Fuller Test:
ADF test statistic       -2.057404
p-value                   0.261984
# lags used              15.000000
# observations          235.000000
critical value (1%)      -3.458487
critical value (5%)      -2.873919
critical value (10%)     -2.573367
Weak evidence against the null hypothesis
Fail to reject the null hypothesis
Data has a unit root and is non-stationary
```

In [15]:

```
#from  first difference (Money) is non- stationary, while (Spending) is now stationary
#Thus we need to apply Second difference
df_D2=df_D1.diff().dropna()
```

In [16]:

```
adf_test(df_D2['Spending'])
print('\n')
```

```
print('\n')
adf_test(df_D2['Money'])
```

```
Augmented Dickey-Fuller Test:
ADF test statistic      -8.760145e+00
p-value                  2.687900e-14
# lags used              8.000000e+00
# observations           2.410000e+02
critical value (1%)     -3.457779e+00
critical value (5%)     -2.873609e+00
critical value (10%)    -2.573202e+00
Strong evidence against the null hypothesis
Reject the null hypothesis
Data has no unit root and is stationary


Augmented Dickey-Fuller Test:
ADF test statistic      -7.077471e+00
p-value                  4.760675e-10
# lags used              1.400000e+01
# observations           2.350000e+02
critical value (1%)     -3.458487e+00
critical value (5%)     -2.873919e+00
critical value (10%)    -2.573367e+00
Strong evidence against the null hypothesis
Reject the null hypothesis
Data has no unit root and is stationary
```

In [17]:

```
# At second differnce,both dataset seem to be stationary,good new.
df_D2.head()
```

Out[17]:

Date	Spending	Money
1995-03-01	35.0	3.7
1995-04-01	-29.8	6.9
1995-05-01	38.1	16.9
1995-06-01	1.5	-0.3
1995-07-01	-51.7	-6.2

In [18]:

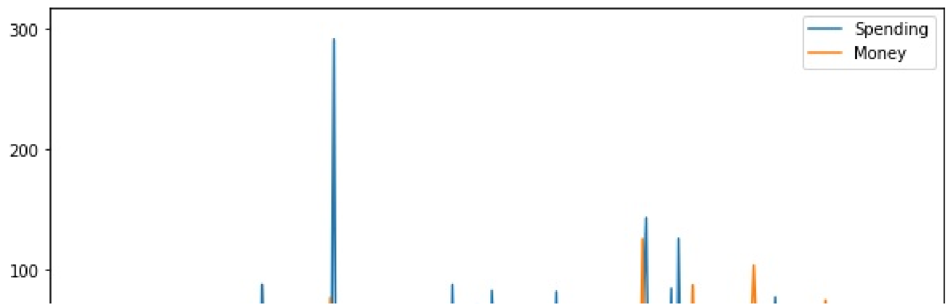
```
# we drop two dates,1 and 2, that while our data starts from the third instead of the first
len(df_D2)
```

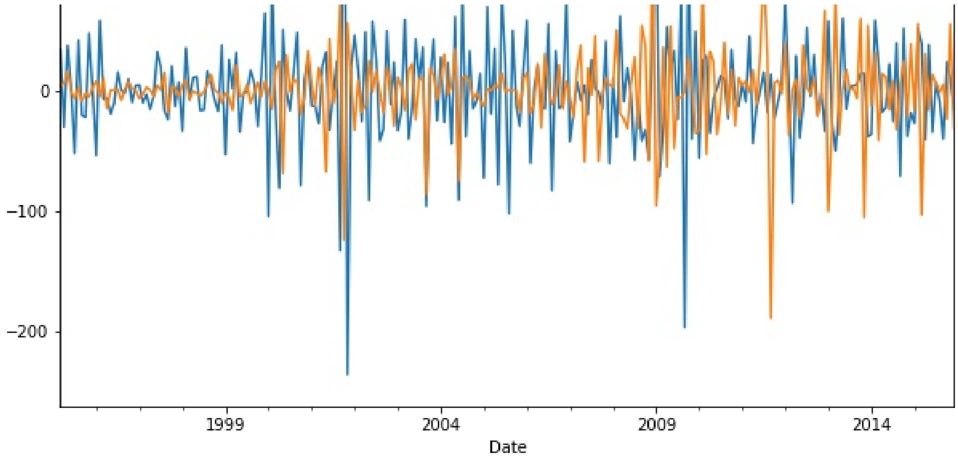
Out[18]:

```
250
```

In [19]:

```
df_D2['Spending'].plot(legend=True,figsize=(10,8))
df_D2['Money'].plot(legend=True);
```





In [20]:

```
#Time to split our Data into Train and Test sets.We are now working with our stationary dataframe of df_D2.
#Let(obs= stands for observations).Lets use 12 months for our test set.
obs=12
train=df_D2[:-obs]
test=df_D2[-obs:]
```

In [21]:

```
print('train',train.shape)

print('test',test.shape)
```

```
train (238, 2)
test (12, 2)
```

In [76]:

```
#Finding the best VAR Model Order,which will be base on the lowest AIC.Lets try 1 to 6
model=VAR(train)
for x in range(7):
    results=model.fit(x)
    print('Order =',x)
    print('AIC',results.aic)
    print('\n')
```

```
Order = 0
AIC 14.747109218090452


Order = 1
AIC 14.178610495220898


Order = 2
AIC 13.955189367163703


Order = 3
AIC 13.849518291541038


Order = 4
AIC 13.827950574458281


Order = 5
AIC 13.78730034460964


Order = 6
AIC 13.799076756885807
```

In [77]:

```
#Order = 5 seems to have the lowest AIC,so lets take it and fit into our model.
results=model.fit(5)
results.summary()
```

Out[77]:

```
  Summary of Regression Results
==================================
Model:                         VAR
Method:                        OLS
Date:           Mon, 07, Sep, 2020
Time:                     01:49:22
--------------------------------------------------------------------
No. of Equations:         2.00000    BIC:                    14.1131
Nobs:                     233.000    HQIC:                   13.9187
Log likelihood:          -2245.45    FPE:                    972321.
AIC:                      13.7873    Det(Omega_mle):         886628.
--------------------------------------------------------------------
Results for equation Spending
==============================================================================
                 coefficient       std. error           t-stat            prob
------------------------------------------------------------------------------
const               0.203469         2.355446            0.086           0.931
L1.Spending        -0.878970         0.067916          -12.942           0.000
L1.Money            0.188105         0.090104            2.088           0.037
L2.Spending        -0.625313         0.090681           -6.896           0.000
L2.Money            0.053017         0.102755            0.516           0.606
L3.Spending        -0.389041         0.098180           -3.963           0.000
L3.Money           -0.022172         0.107057           -0.207           0.836
L4.Spending        -0.245435         0.092069           -2.666           0.008
L4.Money           -0.170456         0.099510           -1.713           0.087
L5.Spending        -0.181699         0.067874           -2.677           0.007
L5.Money           -0.083165         0.088153           -0.943           0.345
==============================================================================

Results for equation Money
==============================================================================
                 coefficient       std. error           t-stat            prob
------------------------------------------------------------------------------
const               0.516683         1.782238            0.290           0.772
L1.Spending        -0.107411         0.051388           -2.090           0.037
L1.Money           -0.646232         0.068177           -9.479           0.000
L2.Spending        -0.192202         0.068613           -2.801           0.005
L2.Money           -0.497482         0.077749           -6.399           0.000
L3.Spending        -0.178099         0.074288           -2.397           0.017
L3.Money           -0.234442         0.081004           -2.894           0.004
L4.Spending        -0.035564         0.069664           -0.511           0.610
L4.Money           -0.295531         0.075294           -3.925           0.000
L5.Spending        -0.058449         0.051357           -1.138           0.255
L5.Money           -0.162399         0.066700           -2.435           0.015
==============================================================================

Correlation matrix of residuals
            Spending     Money
Spending    1.000000 -0.267934
Money      -0.267934  1.000000
```

In [78]:

```
#Predicting the next 12 values,we need to provide date time index manually,VAR.forecast()
#requires we used the order number of previous observation,that is 5.
lagged_value=train.values[-5:]
```

In [79]:

```
train.values[-5:].shape
```

Out[79]:

```
(5, 2)
```

In [60]:

```
# 5 is the lagged order
# 2 meaning, we are dealing with two time series
```

In [80]:

```
z=results.forecast(y=train.values[-5:],steps=12)
```

In [81]:

```
z
```

Out[81]:

```
array([[ 36.14982003, -16.99527634],
       [-11.45029844,  -3.17403756],
       [ -6.68496939,  -0.377725  ],
       [  5.47945777,  -2.60223305],
       [ -2.44336505,   4.228557  ],
       [  0.38763902,   1.55939341],
       [  3.88368011,  -0.99841027],
       [ -2.3561014 ,   0.36451042],
       [ -1.22414652,  -1.21062726],
       [  0.786927  ,   0.22587712],
       [  0.18097449,   1.33893884],
       [  0.21275046,  -0.21858453]])
```

In [82]:

```
test
```

Out[82]:

	Spending	Money
Date		
2015-01-01	-26.6	-15.5
2015-02-01	52.4	56.1
2015-03-01	39.5	-102.8
2015-04-01	-40.4	30.9
2015-05-01	38.8	-15.8
2015-06-01	-34.1	14.0
2015-07-01	6.9	6.7
2015-08-01	-8.5	-0.7
2015-09-01	-39.8	5.5
2015-10-01	24.5	-23.1
2015-11-01	10.7	55.8
2015-12-01	-15.0	-31.2

In [83]:

```
#lets tranform it back to dataframe for easy understanding.
idx=pd.date_range('2015-01-01',periods=12,freq='MS')
```

In [84]:

```
Forecast=pd.DataFrame(data=z,index=idx,columns=['Spending_2d','Money_2d',])
Forecast
```

Out[84]:

	Spending_2d	Money_2d
2015-01-01	36.149820	-16.995276
2015-02-01	-11.450298	-3.174038
2015-03-01	-6.684969	-0.377725
2015-04-01	5.479458	-2.602233
2015-05-01	-2.443365	4.228557
2015-06-01	0.387639	1.559393
2015-07-01	3.883680	-0.998410
2015-08-01	-2.356101	0.364510
2015-09-01	-1.224147	-1.210627
2015-10-01	0.786927	0.225877
2015-11-01	0.180974	1.338939
2015-12-01	0.212750	-0.218585

In [85]:

```
#we have some difficulties here,we have to invert the transformation to compare with the original data,
#remember our forecasted values represent second-order diff
```

In [86]:

```
# Add the most recent first difference from the training side of the original dataset to the forecast cumulative sum
Forecast['Money_1d'] = (df['Money'].iloc[-obs-1]-df['Money'].iloc[-obs-2]) + Forecast['Money_2d'].cumsum()

# Now build the forecast values from the first difference set
Forecast['MoneyForecast'] = df['Money'].iloc[-obs-1] + Forecast['Money_1d'].cumsum()
```

In [87]:

```
# Add the most recent first difference from the training side of the original dataset to the forecast cumulative sum
Forecast['Spending_1d'] = (df['Spending'].iloc[-obs-1]-df['Spending'].iloc[-obs-2]) + Forecast['Spending_2d'].cumsum()

# Now build the forecast values from the first difference set
Forecast['SpendingForecast'] = df['Spending'].iloc[-obs-1] + Forecast['Spending_1d'].cumsum()
```

In [88]:

```
Forecast
```

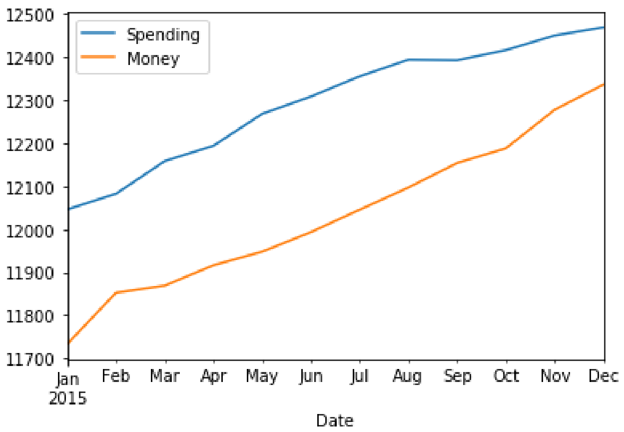
Out[88]:

	Spending_2d	Money_2d	Money_1d	MoneyForecast	Spending_1d	SpendingForecast
2015-01-01	36.149820	-16.995276	61.604724	11731.704724	46.749820	12108.749820
2015-02-01	-11.450298	-3.174038	58.430686	11790.135410	35.299522	12144.049342
2015-03-01	-6.684969	-0.377725	58.052961	11848.188371	28.614552	12172.663894
2015-04-01	5.479458	-2.602233	55.450728	11903.639099	34.094010	12206.757904
2015-05-01	-2.443365	4.228557	59.679285	11963.318384	31.650645	12238.408549
2015-06-01	0.387639	1.559393	61.238678	12024.557062	32.038284	12270.446833
2015-07-01	3.883680	-0.998410	60.240268	12084.797331	35.921964	12306.368797
2015-08-01	-2.356101	0.364510	60.604779	12145.402109	33.565863	12339.934659
2015-09-01	-1.224147	-1.210627	59.394151	12204.796261	32.341716	12372.276375
2015-10-01	0.786927	0.225877	59.620028	12264.416289	33.128643	12405.405019

	Spending_2d	Money_2d	Money_1d	MoneyForecast	Spending_1d	SpendingForecast
2015-11-01	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
2015-12-01	0.212750	-0.218585	60.740383	12386.115639	33.522368	12472.237004

In [89]:

```
# lets plot our forcast,remember we have to use out true test that is from 2015-01-01 that 12 and
above
test_range=df[-obs:]
test_range.plot();
```



In [90]:

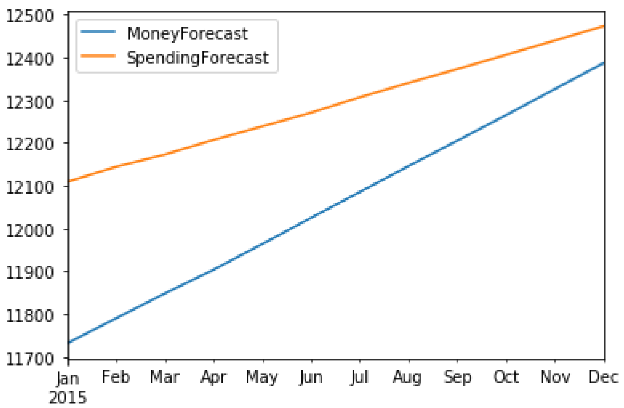
```
Forecast.columns
```

Out[90]:

```
Index(['Spending_2d', 'Money_2d', 'Money_1d', 'MoneyForecast', 'Spending_1d',
       'SpendingForecast'],
      dtype='object')
```

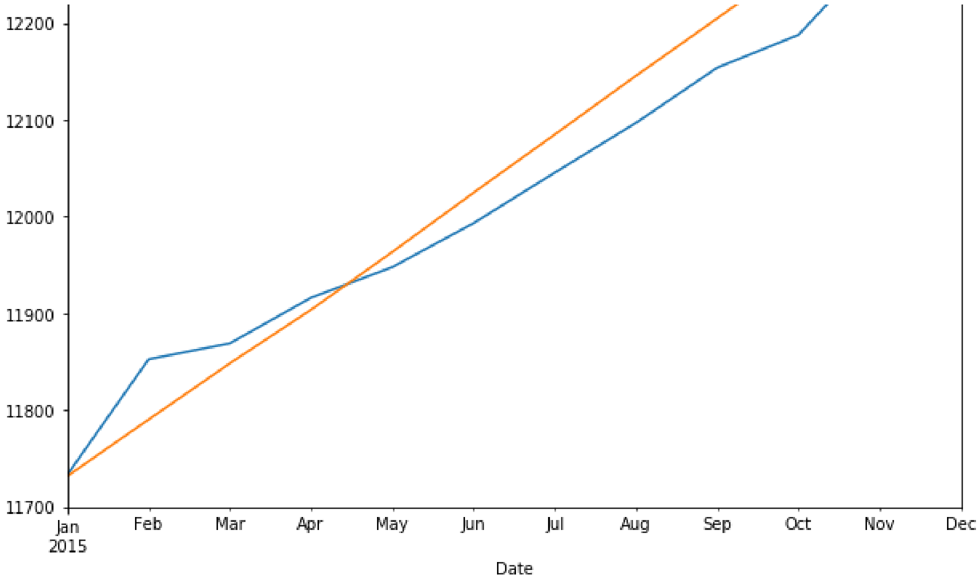
In [91]:

```
Forecast[['MoneyForecast','SpendingForecast',]].plot();
```



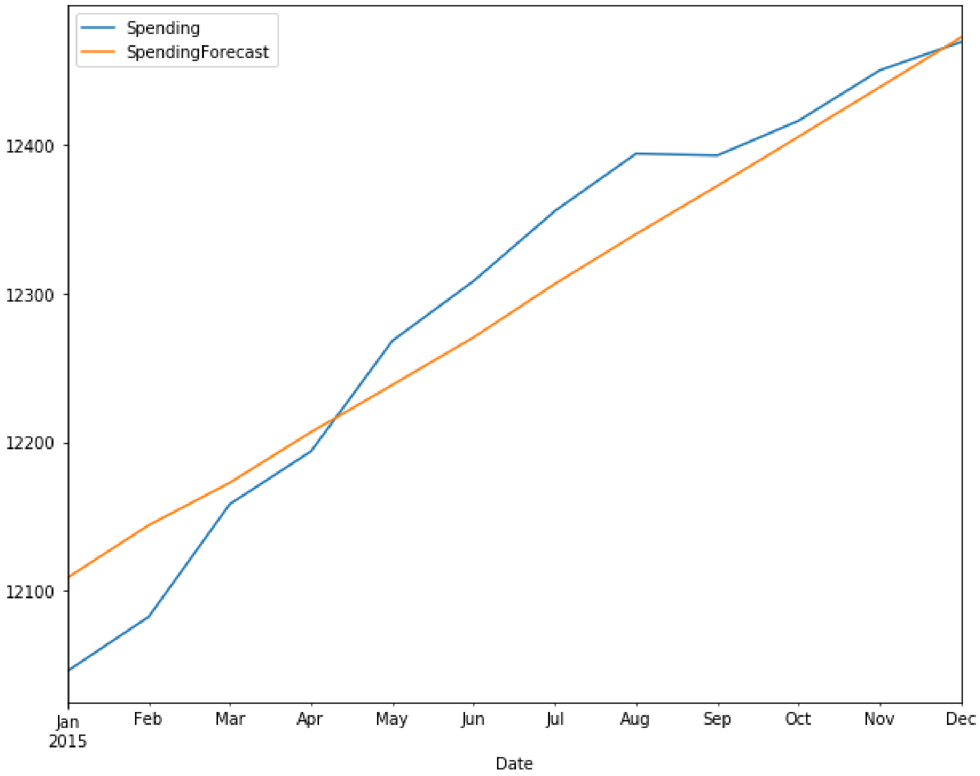
In [92]:

```
test_range['Money'].plot(legend=True,figsize=(10,8))
Forecast['MoneyForecast'].plot(legend=True);
```



In [93]:

```
test_range['Spending'].plot(legend=True,figsize=(10,8))
Forecast['SpendingForecast'].plot(legend=True);
```



In [94]:

```
rmse(test_range['Money'],Forecast['MoneyForecast'],)
```

Out[94]:

```
43.71049653558893
```

In [95]:

```
test_range['Money'].mean()
```

Out[95]:

```
12034.008333333333
```

In [97]:

```
rmse(test_range['Spending'],Forecast['SpendingForecast'])
```

Out[97]:

```
37.001175169408086
```

In [98]:

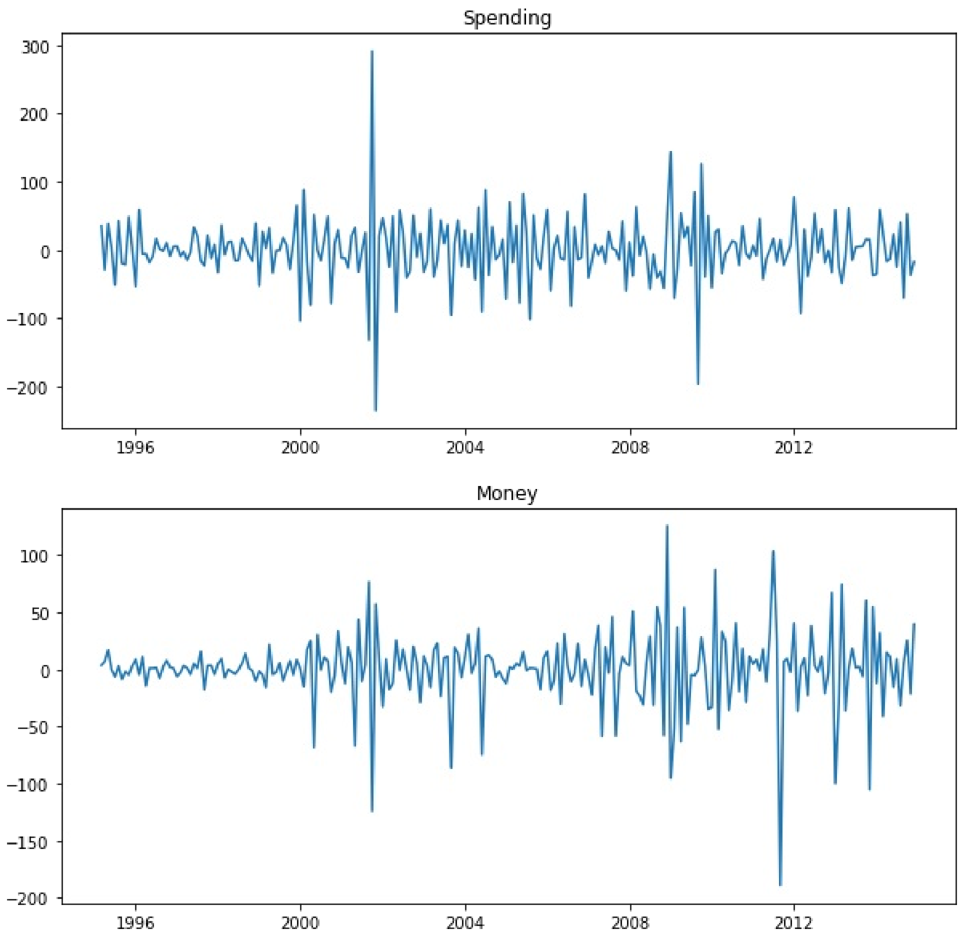
```
test_range['Spending'].mean()
```

Out[98]:

```
12294.533333333333
```

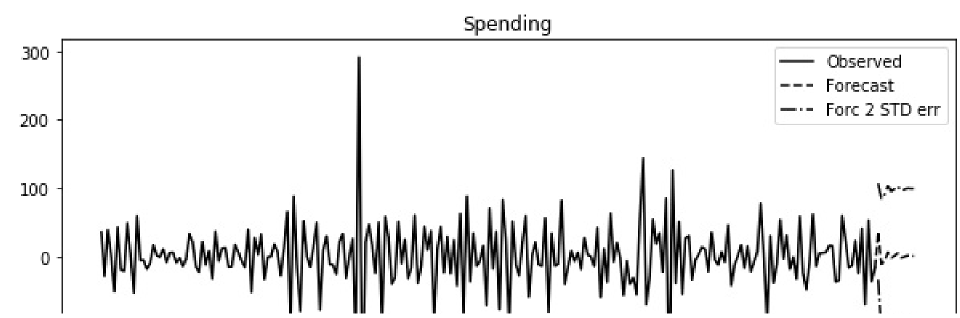
In [100]:

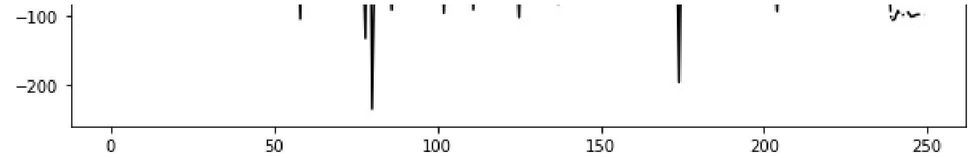
```
results.plot();
```

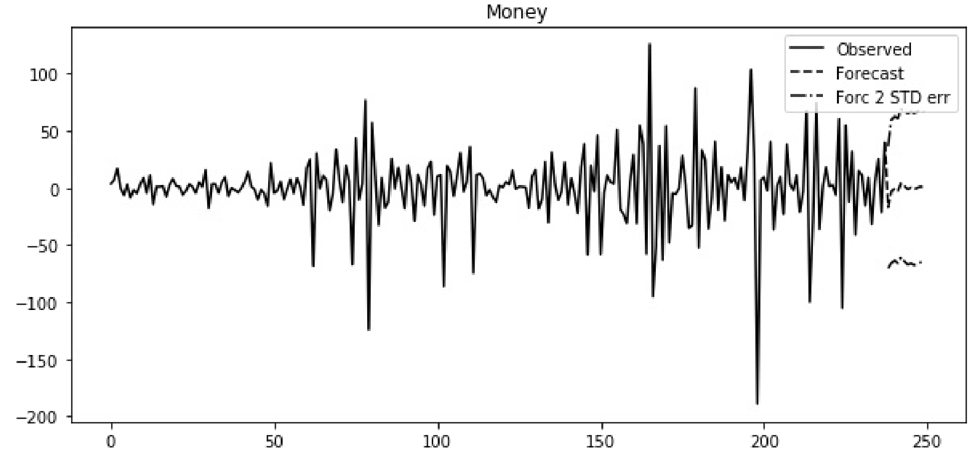


In [107]:

```
results.plot_forecast(12);
```







In [108]:

```
#lets compare our results with AR(5)
from statsmodels.tsa.ar_model import AR,ARResults
```

In [109]:

```
# lets begin with money
model_5=AR(train['Money'])
```

In [113]:

```
AR5=model_5.fit(maxlag=5,method='mle')
```

In [111]:

```
AR5.params
```

Out[111]:

```
const        0.585208
L1.Money    -0.605217
L2.Money    -0.465398
L3.Money    -0.228645
L4.Money    -0.311355
L5.Money    -0.127613
dtype: float64
```

In [114]:

```
start=len(train)
end=len(train)+len(test)-1
z1=pd.DataFrame(AR5.predict(start=start,end=end,dynamic=False),columns=['Money'])
z1
```

Out[114]:

	Money
2015-01-01	-16.911056
2015-02-01	-11.347193
2015-03-01	9.669332
2015-04-01	-5.699593
2015-05-01	2.353698

	Money
2015-06-01	5.293522
2015-07-01	-3.973283
2015-08-01	0.528810
2015-09-01	0.898493
2015-10-01	-1.244737
2015-11-01	1.361054
2015-12-01	0.477734

In [116]:

```
# Add the most recent first difference from the training side of the original dataset to the forec
ast cumulative sum
z1['Money_1d'] = (df['Money'].iloc[-obs-1]-df['Money'].iloc[-obs-2]) + z1['Money'].cumsum()

# Now build the forecast values from the first difference set
z1['MoneyForecast'] = df['Money'].iloc[-obs-1] + z1['Money_1d'].cumsum()
```

In [117]:

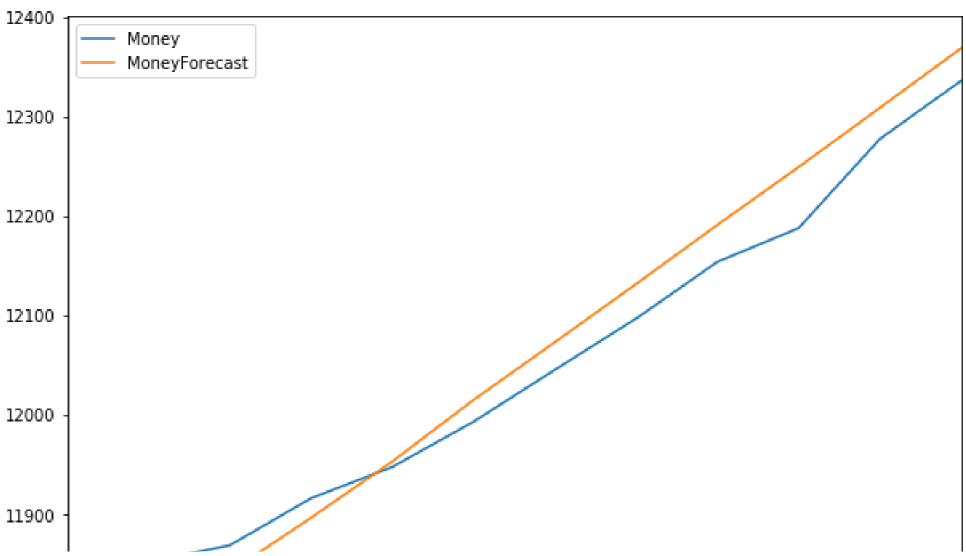
```
z1
```

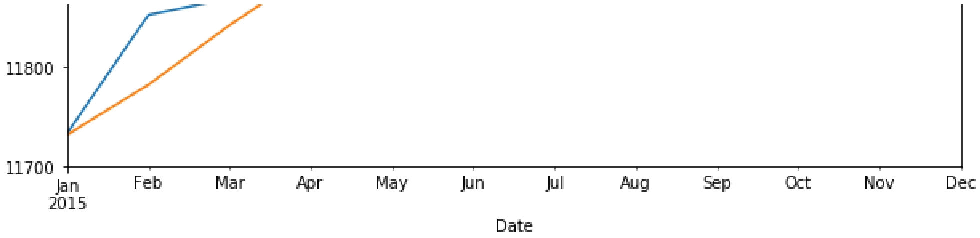
Out[117]:

	Money	Money_1d	MoneyForecast
2015-01-01	-16.911056	61.688944	11731.788944
2015-02-01	-11.347193	50.341751	11782.130695
2015-03-01	9.669332	60.011083	11842.141778
2015-04-01	-5.699593	54.311490	11896.453268
2015-05-01	2.353698	56.665188	11953.118456
2015-06-01	5.293522	61.958710	12015.077167
2015-07-01	-3.973283	57.985427	12073.062594
2015-08-01	0.528810	58.514237	12131.576830
2015-09-01	0.898493	59.412730	12190.989560
2015-10-01	-1.244737	58.167993	12249.157553
2015-11-01	1.361054	59.529046	12308.686599
2015-12-01	0.477734	60.006780	12368.693379

In [118]:

```
test_range['Money'].plot(legend=True,figsize=(10,8))
z1['MoneyForecast'].plot(legend=True);
```





In [119]:

```
rmse(test_range['Money'],z1['MoneyForecast'])
```

Out[119]:

36.22201359217574

In [135]:

```
test_range['Money'].mean()
```

Out[135]:

12034.008333333333

In [123]:

```
#lets move to spending
model_5=AR(train['Spending'])
```

In [124]:

```
ARs5=model_5.fit(maxlag=5,method='mle')
```

In [125]:

```
ARs5.params
```

Out[125]:

```
const          0.221066
L1.Spending   -0.913123
L2.Spending   -0.677036
L3.Spending   -0.450798
L4.Spending   -0.273218
L5.Spending   -0.159474
dtype: float64
```

In [126]:

```
z2=pd.DataFrame(ARs5.predict(start=start,end=end,dynamic=False),columns=['Spending'])
z2
```

Out[126]:

	Spending
2015-01-01	30.883255
2015-02-01	-2.227389
2015-03-01	-8.838589
2015-04-01	6.673427
2015-05-01	-4.483686
2015-06-01	-0.535024
2015-07-01	3.506935
2015-08-01	-1.011510

	Spending
2015-09-01	-0.827647
2015-10-01	0.941930
2015-11-01	-0.495535
2015-12-01	0.126030

In [127]:

```
# Add the most recent first difference from the training side of the original dataset to the forec
ast cumulative sum
z2['Spending_1d'] = (df['Spending'].iloc[-obs-1]-df['Spending'].iloc[-obs-2]) + z2['Spending'].cums
um()

# Now build the forecast values from the first difference set
z2['SpendingForecast'] = df['Spending'].iloc[-obs-1] + z2['Spending_1d'].cumsum()
```

In [128]:

```
z2
```

Out[128]:

	Spending	Spending_1d	SpendingForecast
2015-01-01	30.883255	41.483255	12103.483255
2015-02-01	-2.227389	39.255866	12142.739121
2015-03-01	-8.838589	30.417277	12173.156398
2015-04-01	6.673427	37.090705	12210.247103
2015-05-01	-4.483686	32.607019	12242.854121
2015-06-01	-0.535024	32.071995	12274.926116
2015-07-01	3.506935	35.578930	12310.505046
2015-08-01	-1.011510	34.567420	12345.072466
2015-09-01	-0.827647	33.739773	12378.812239
2015-10-01	0.941930	34.681703	12413.493941
2015-11-01	-0.495535	34.186167	12447.680109
2015-12-01	0.126030	34.312197	12481.992306

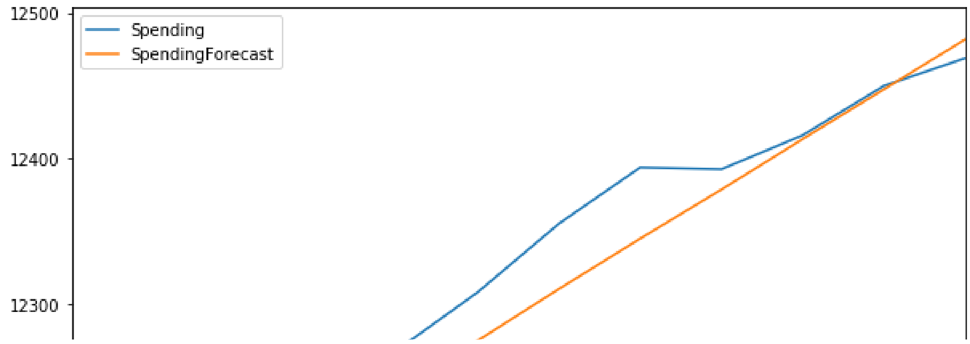
In [136]:

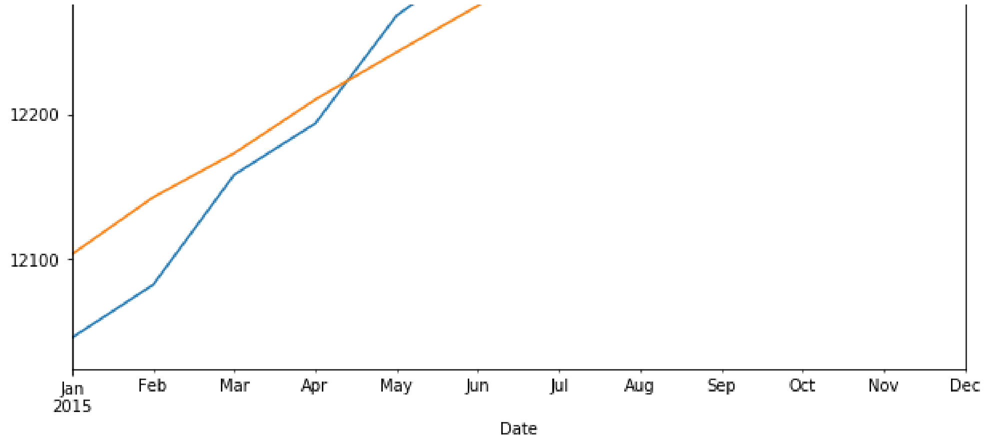
```
print('rmse:',rmse(test_range['Spending'],z2['SpendingForecast']))
print('mean_test:',test_range['Money'].mean())
```

```
rmse: 34.121719997216175
mean_test: 12034.008333333333
```

In [137]:

```
test_range['Spending'].plot(legend=True,figsize=(10,8))
z2['SpendingForecast'].plot(legend=True);
```





It looks like AR(5) was able to perform more than VAR(5).Our conclusion was base on error metric using the rmse.AR(5) comes out with smaller rmse than VAR(5) for both spendings and money.VARMA was also been used in this project but did perform poor.Here we did use the auto_arima to come out with the best order,(1,1,2) for spending and (1,2,2) for money.ARMA model was also test but nothing much came out of it.ARMA(1,2).

In [143]:

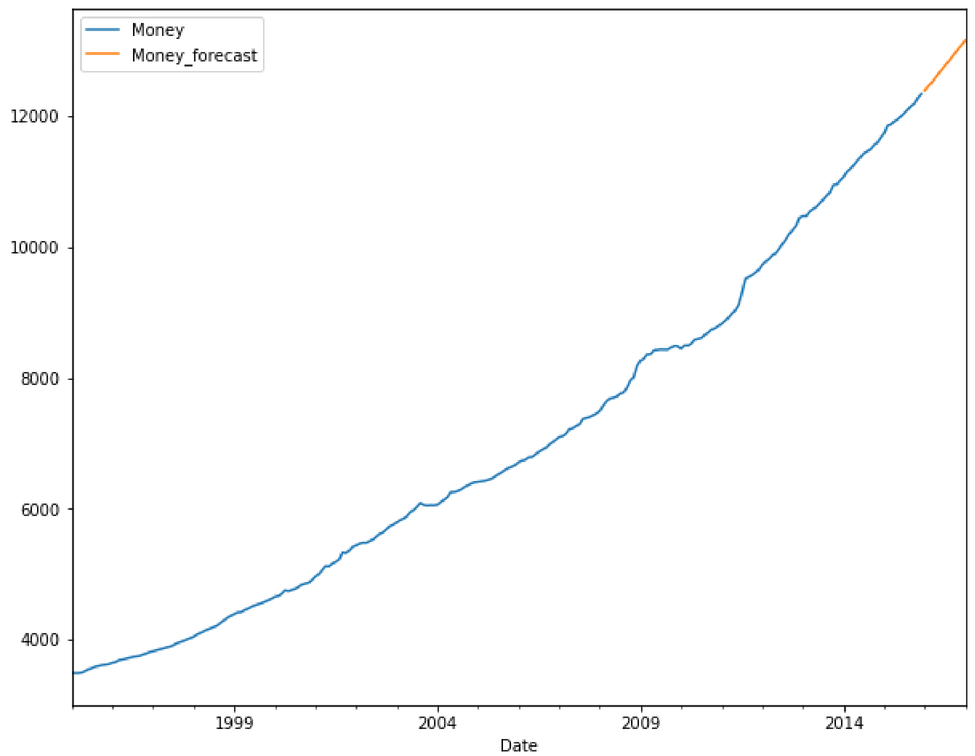
```
#lets predict money and spending 12months into the future
model3=AR(df['Money'])
ar_fit=model3.fit()
```

In [144]:

```
forecasting=ar_fit.predict(start=len(df['Money']),end=len(df['Money'])+12).rename('Money_forecast')
```

In [145]:

```
df['Money'].plot(legend=True,figsize=(10,8))
forecasting.plot(legend=True);
```



In [146]:

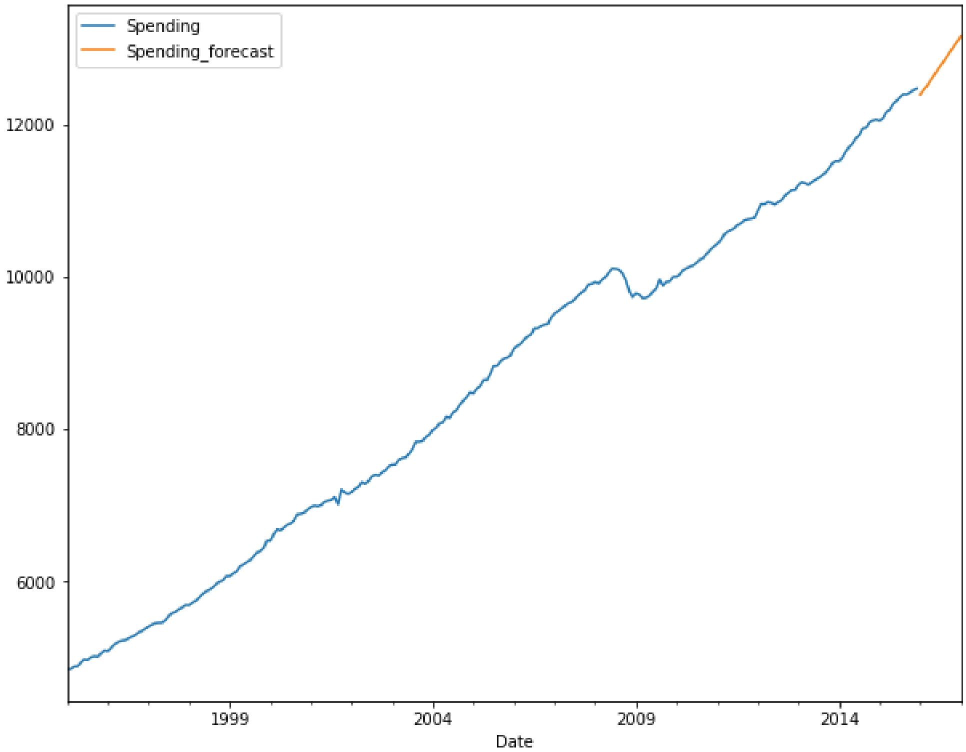
```
model4=AR(df['Spending'])
ar1_fit=model4.fit()
```

In [147]:

```
forecasting1=ar_fit.predict(start=len(df['Spending']),end=len(df['Spending'])+12).rename('Spending_
forecast')
```

In [148]:

```
df['Spending'].plot(legend=True,figsize=(10,8))
forecasting1.plot(legend=True);
```



In [ ]: