machine-learning-assignment-7

December 8, 2023

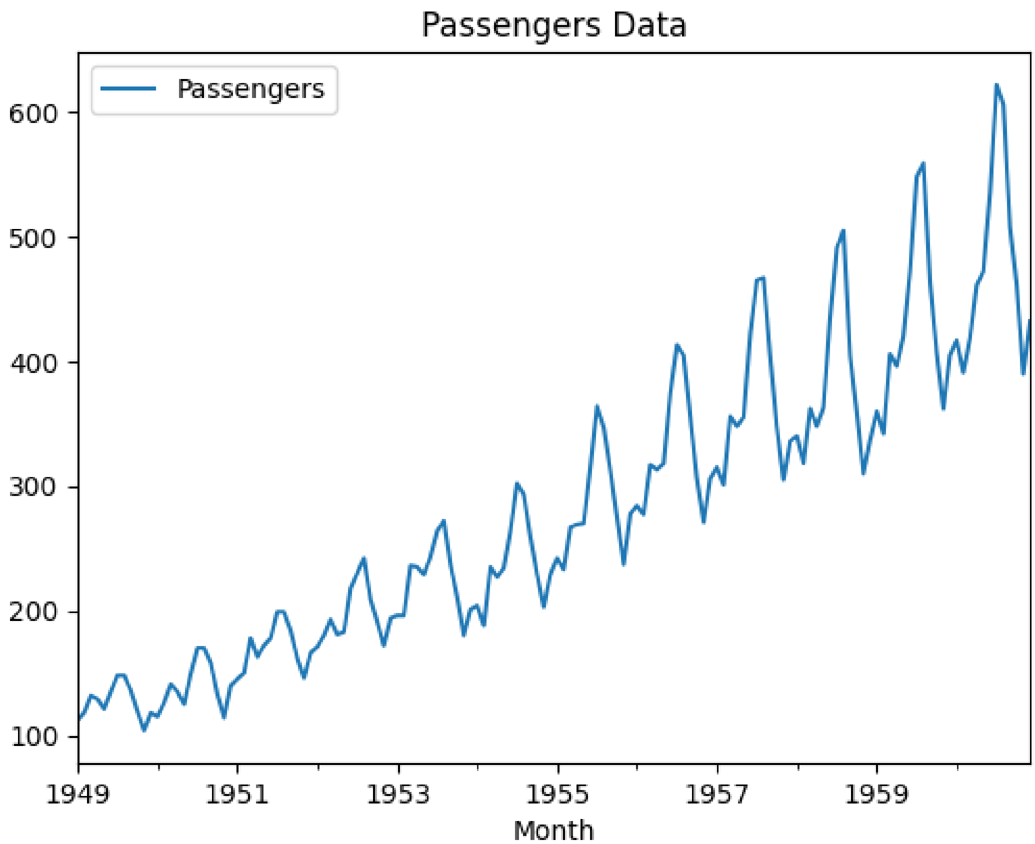
Q1. Implement Holt-winters exponential smoothing on air-passengers dataset and observe the metrics difference in them.

[8]:
```
# dataframe opertations - pandas
import pandas as pd
# plotting data - matplotlib
from matplotlib import pyplot as plt
# time series - statsmodels
# Seasonality decomposition
from statsmodels.tsa.seasonal import seasonal_decompose
# holt winters
# single exponential smoothing
from statsmodels.tsa.holtwinters import SimpleExpSmoothing
# double and triple exponential smoothing
from statsmodels.tsa.holtwinters import ExponentialSmoothing
```

[9]:
```
airline = pd.read_csv('airline_passengers.
  ↪csv',index_col='Month',parse_dates=True)
# finding shape of the dataframe
print(airline.shape)
# having a look at the data
print(airline.head())
# plotting the original data
airline[['Passengers']].plot(title='Passengers Data')
```

```
(144, 1)
            Passengers
Month
1949-01-01         112
1949-02-01         118
1949-03-01         132
1949-04-01         129
1949-05-01         121
```

[9]:
```
<Axes: title={'center': 'Passengers Data'}, xlabel='Month'>
```

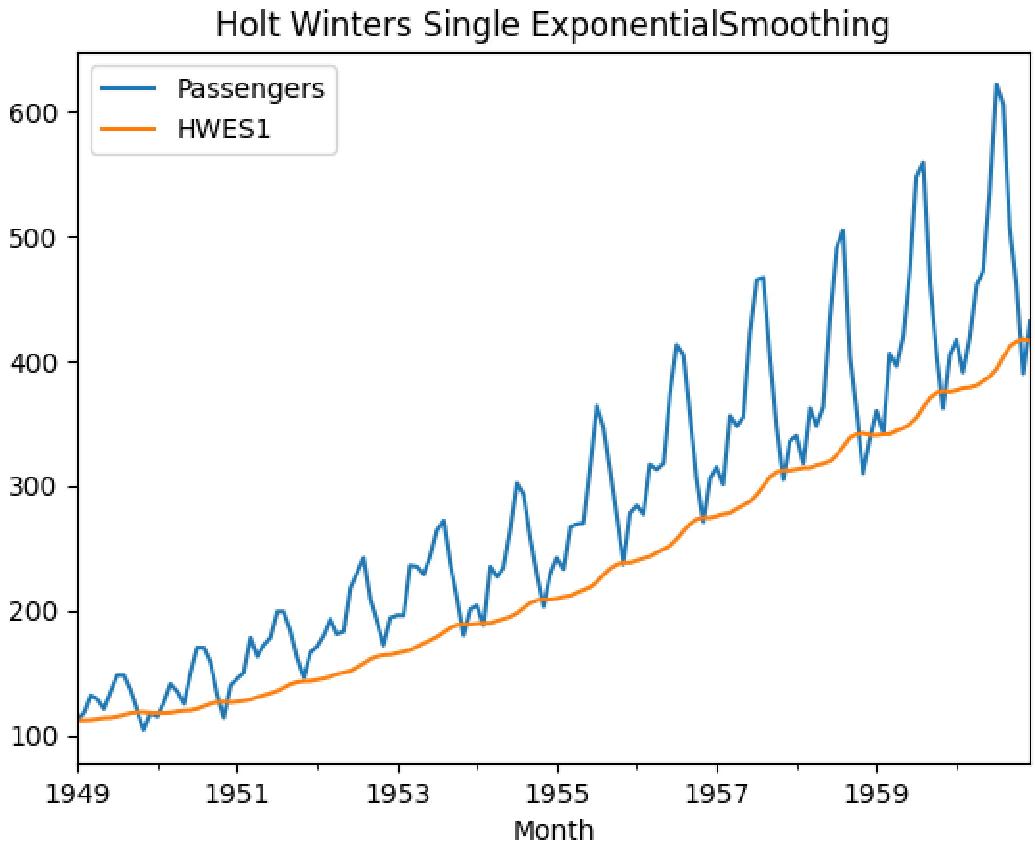


[10]:
```
# Set the frequency of the date time index as Monthly start as indicated by the␣
↪data
airline.index.freq = 'MS'
# Set the value of Alpha and define m (Time Period)
m = 12
alpha = 1/(2*m)
alpha
```

[10]: 0.041666666666666664

[11]:
```
airline['HWES1'] = SimpleExpSmoothing(airline['Passengers']).
↪fit(smoothing_level=alpha,optimized=False,use_brute=True).fittedvalues
airline[['Passengers','HWES1']].plot(title='Holt Winters Single␣
↪ExponentialSmoothing')
```

[11]:
```
<Axes: title={'center': 'Holt Winters Single ExponentialSmoothing'},
xlabel='Month'>
```



[63]:
```
tlen=120
train=airline[0:tlen]
test=airline[tlen:]
print(train.shape)
```

```
(120, 2)
```

[64]:
```
print(test.shape)
```

```
(24, 2)
```

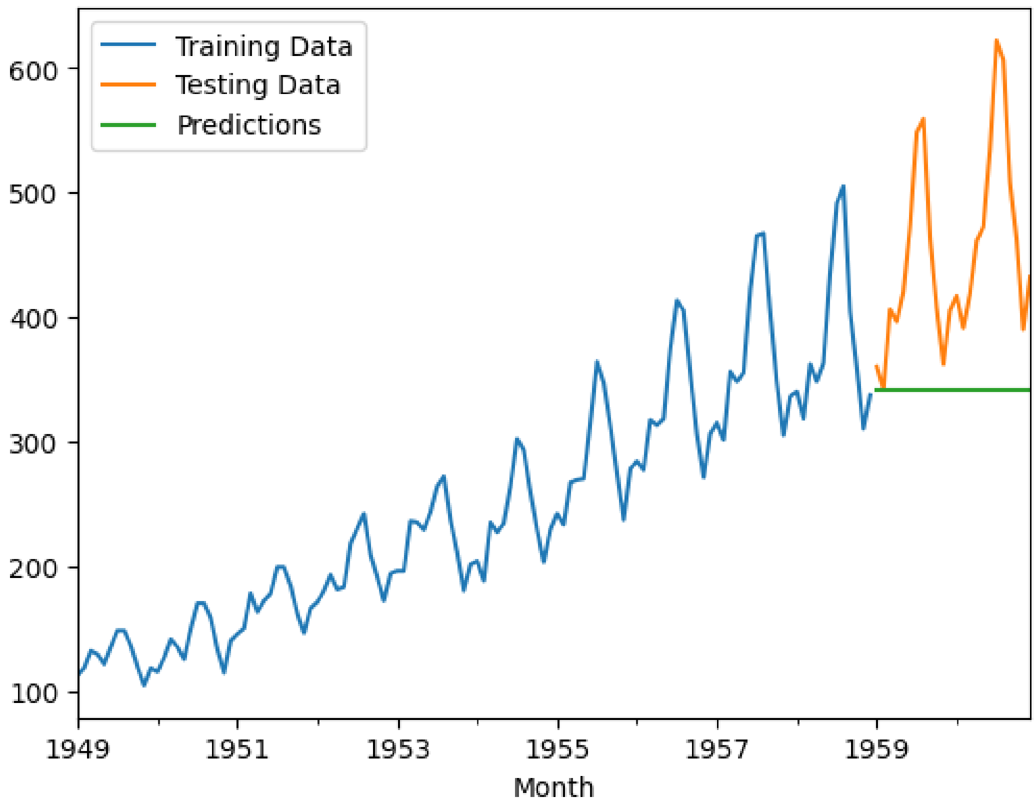
[66]:
```
#Forecasting for the test data
#Developing model
sexpsmooth_model=SimpleExpSmoothing(train['Passengers']).
 ↪fit(smoothing_level=alpha,optimized=False,use_brute=True)
```

[67]:
```
test_pred1=sexpsmooth_model.forecast(24)
```

[68]:
```
train['Passengers'].plot(label='Training Data')
test['Passengers'].plot(label='Testing Data')
```

```
test_pred1.plot(label='Predictions')
plt.legend(loc='best')
```

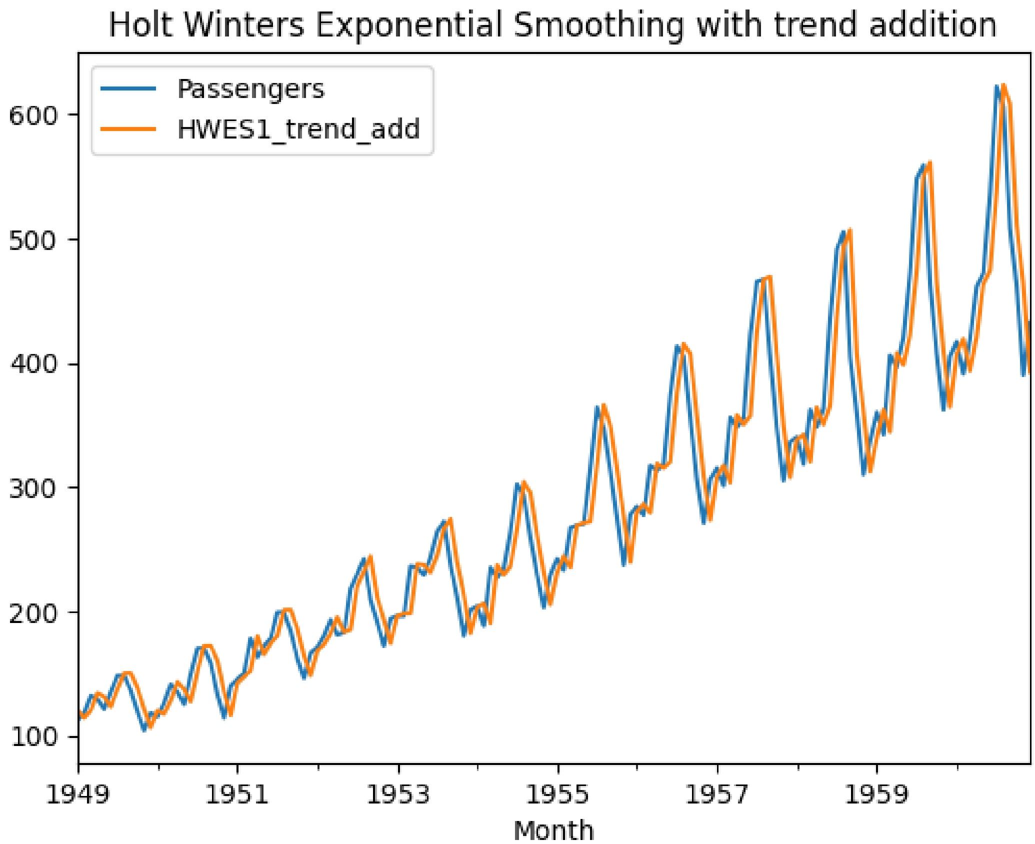
[68]: <matplotlib.legend.Legend at 0x78e9099b0df0>



[69]:
```
#Finding trend with holt-linear trend methods
airline['HWES1_trend_add']=ExponentialSmoothing(airline['Passengers'],trend='add').
↪fit().fittedvalues
```

[70]:
```
airline[['Passengers','HWES1_trend_add']].plot(title='Holt Winters Exponential␣
↪Smoothing with trend addition')
```

[70]: <Axes: title={'center': 'Holt Winters Exponential Smoothing with trend addition'}, xlabel='Month'>

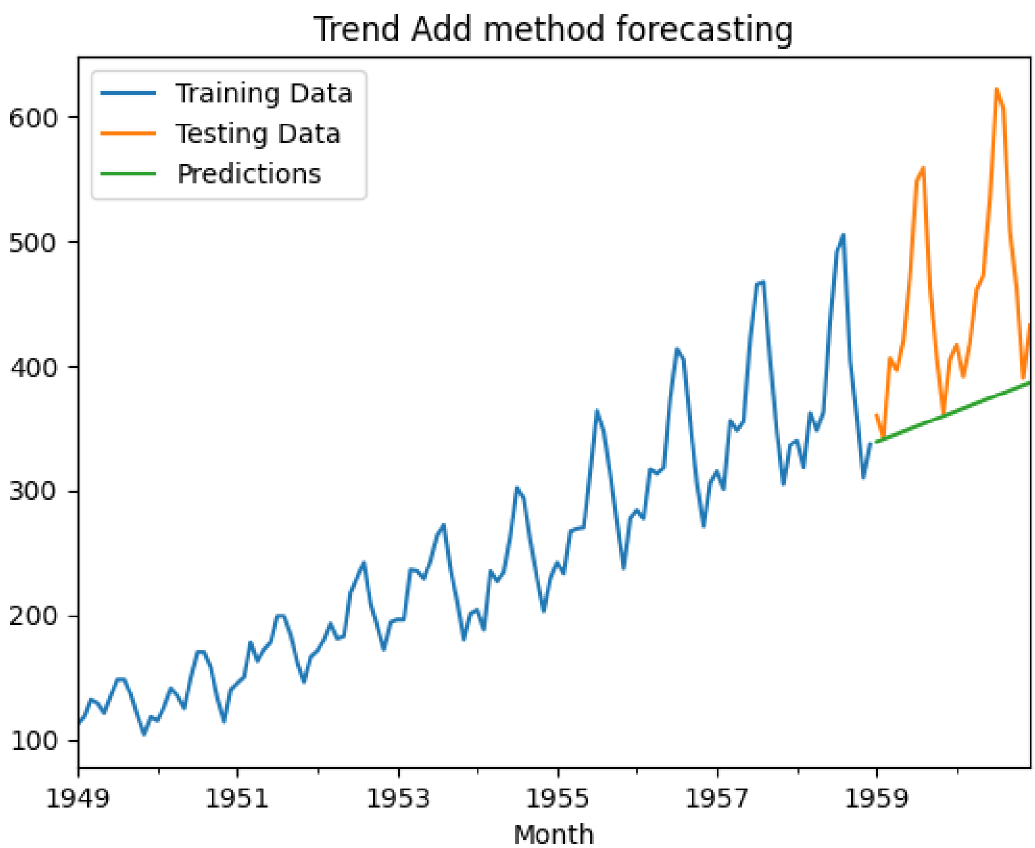


[71]:
```
#Forecasting with trend add method
exp_trend_add_model=ExponentialSmoothing(train['Passengers'],trend='add').fit()
```

[72]:
```
test_pred2=exp_trend_add_model.forecast(24)
```

[73]:
```
train['Passengers'].plot(label='Training Data')
test['Passengers'].plot(label='Testing Data')
test_pred2.plot(label='Predictions')
plt.legend(loc='best')
plt.title("Trend Add method forecasting")
```

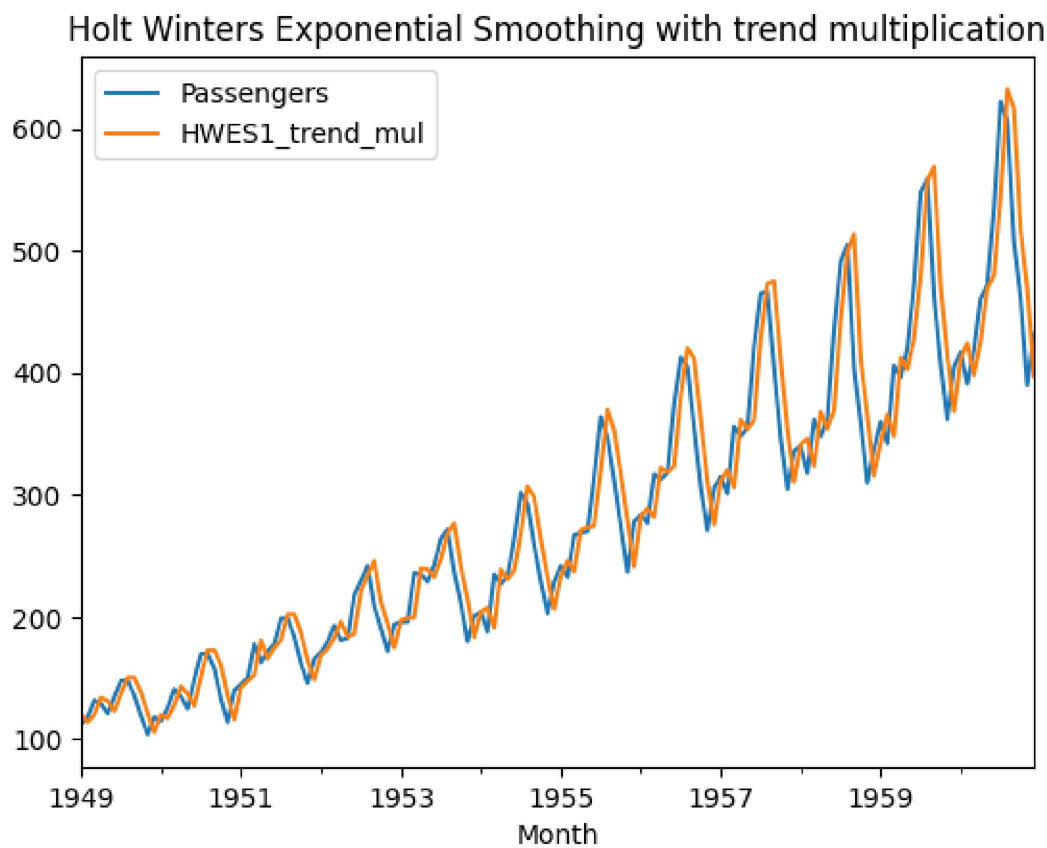
[73]: Text(0.5, 1.0, 'Trend Add method forecasting')



[74]:
```
#Plotting trend with trend multiplication method
airline['HWES1_trend_mul']=ExponentialSmoothing(airline['Passengers'],trend='mul').
    ↪fit().fittedvalues
```

[75]:
```
airline[['Passengers','HWES1_trend_mul']].plot(title='Holt Winters Exponential␣
    ↪Smoothing with trend multiplication')
```

[75]:
```
<Axes: title={'center': 'Holt Winters Exponential Smoothing with trend
multiplication'}, xlabel='Month'>
```



Q2.Explain ARMA and ARIMA model, what is purpose of these models in time series and Explain difference between them.

Ans= ARMA (AutoRegressive Moving Average) and ARIMA (AutoRegressive Integrated Moving Average) are models used in time series analysis to forecast future values based on past observations.

ARMA Model:

```
Combines autoregression (AR) and moving average (MA) components.
AR Component: Current value depends on its own past values (autocorrelation).
MA Component: Current value depends on the average of past error terms.
Denoted as ARMA(p, q), where 'p' represents AR order, and 'q' represents MA order.
Assumes the time series data is stationary (constant mean, variance).
```

ARIMA Model:

```
Includes autoregressive (AR) and moving average (MA) components with an additional integration
AR and MA Components: Similar to ARMA.
Integration Component: Involves differencing the original data to make it stationary.
Denoted as ARIMA(p, d, q), where 'p' represents AR order, 'd' represents differencing order, an
Suitable for handling non-stationary data by differencing to achieve stationarity.
```

Q3.Implement different ARIMA models on shampoo sales dataset. Display the sum-

mary of each model, and observe the metrics difference in them.

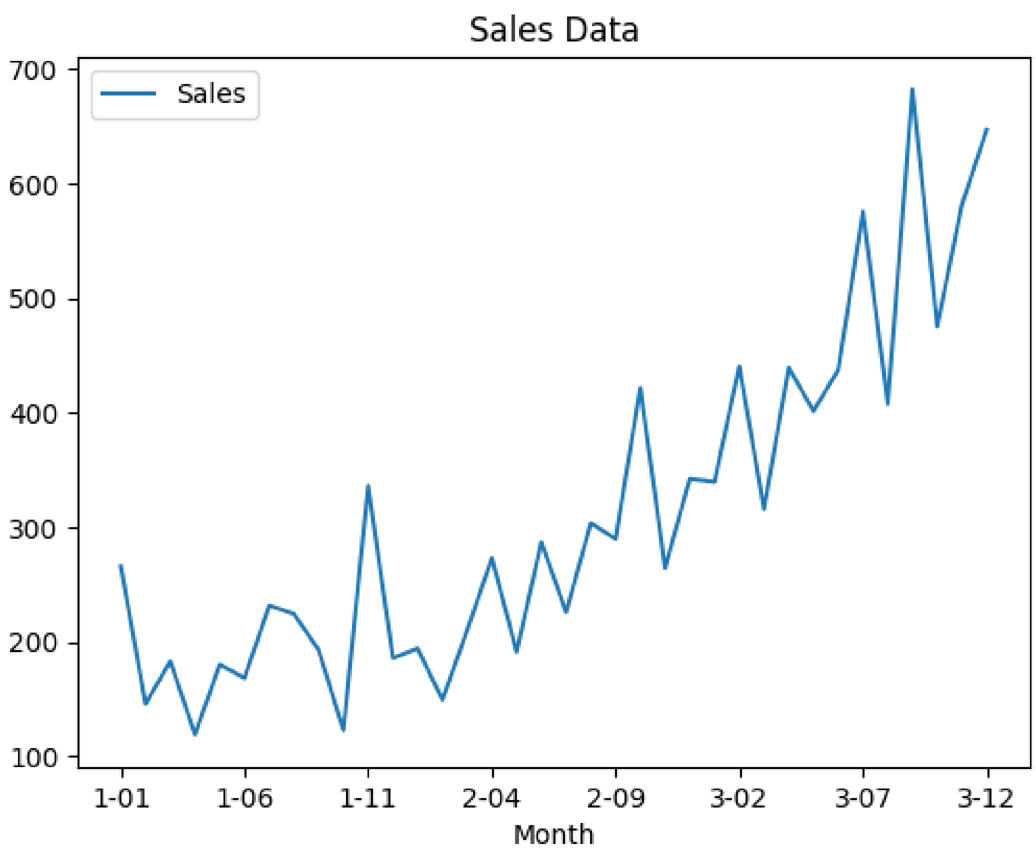
[2]:
```python
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.arima.model import ARIMA
```

[3]:
```python
shampoo = pd.read_csv('shampoo.csv',index_col='Month',parse_dates=True )
 shampoo.head()
```

[3]:
```
       Sales
Month
1-01   266.0
1-02   145.9
1-03   183.1
1-04   119.3
1-05   180.3
```

[4]:
```python
# plotting the original data
shampoo[['Sales']].plot(title='Sales Data')
```

[4]:
```
<Axes: title={'center': 'Sales Data'}, xlabel='Month'>
```



[16]:
```
tlen=12
train=shampoo[0:tlen]
test=shampoo[tlen:]
```

[17]:
```
test
```

[17]:

Month	Sales
2-01	194.3
2-02	149.5
2-03	210.1
2-04	273.3
2-05	191.4
2-06	287.0
2-07	226.0
2-08	303.6
2-09	289.9
2-10	421.6
2-11	264.5
2-12	342.3
3-01	339.7
3-02	440.4
3-03	315.9
3-04	439.3
3-05	401.3
3-06	437.4
3-07	575.5
3-08	407.6
3-09	682.0
3-10	475.3
3-11	581.3
3-12	646.9

[18]:
```
arima_model1=ARIMA(train,order=(1,0,0)).fit()
arima_model2=ARIMA(train,order=(0,1,0)).fit()
arima_model3=ARIMA(train,order=(0,0,1)).fit()
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
ValueWarning: An unsupported index was provided and will be ignored when e.g.
forecasting.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
ValueWarning: An unsupported index was provided and will be ignored when e.g.
forecasting.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
```

```
ValueWarning: An unsupported index was provided and will be ignored when e.g.
forecasting.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
ValueWarning: An unsupported index was provided and will be ignored when e.g.
forecasting.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
ValueWarning: An unsupported index was provided and will be ignored when e.g.
forecasting.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
ValueWarning: An unsupported index was provided and will be ignored when e.g.
forecasting.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
ValueWarning: An unsupported index was provided and will be ignored when e.g.
forecasting.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
ValueWarning: An unsupported index was provided and will be ignored when e.g.
forecasting.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
ValueWarning: An unsupported index was provided and will be ignored when e.g.
forecasting.
  self._init_dates(dates, freq)
```

[19]:

```
pred1=arima_model1.forecast(12)
pred2=arima_model2.forecast(12)
pred3=arima_model3.forecast(12)
```

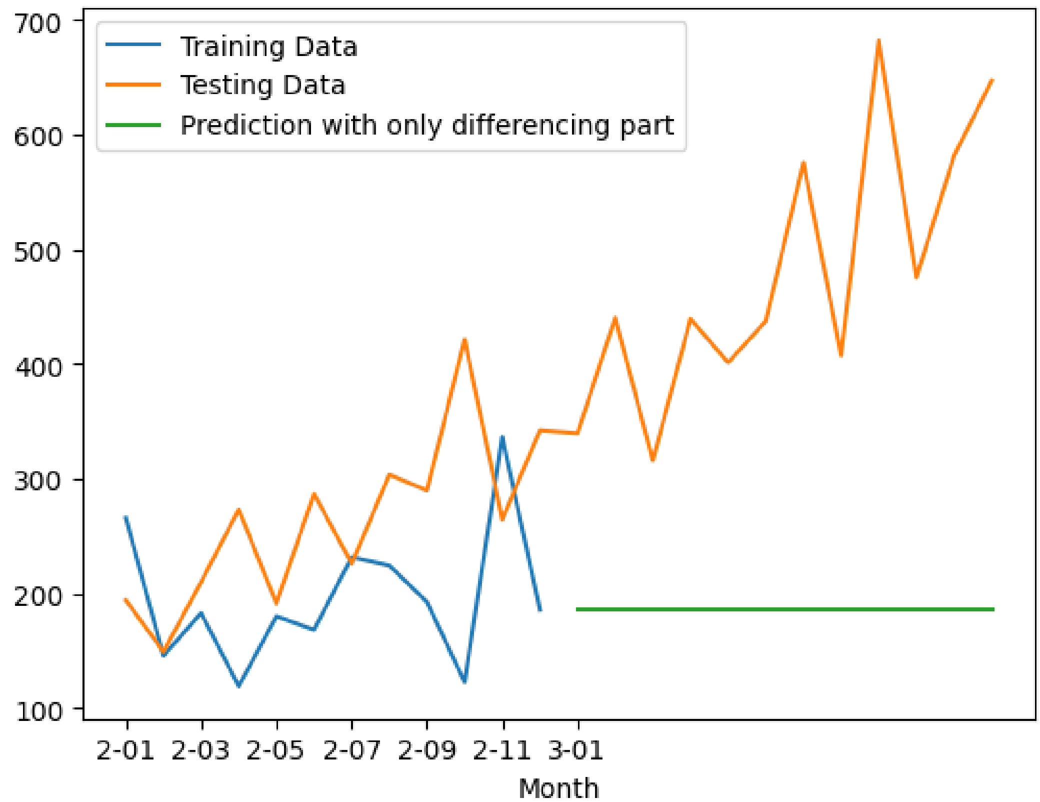
```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:836:
ValueWarning: No supported index is available. Prediction results will be given
with an integer index beginning at `start`.
  return get_prediction_index(
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:836:
FutureWarning: No supported index is available. In the next version, calling
this method in a model without a supported index will result in an exception.
  return get_prediction_index(
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:836:
ValueWarning: No supported index is available. Prediction results will be given
with an integer index beginning at `start`.
  return get_prediction_index(
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:836:
ValueWarning: No supported index is available. Prediction results will be given
with an integer index beginning at `start`.
  return get_prediction_index(
```

[20]:
```
train['Sales'].plot(label="Sales Data")
test['Sales'].plot(label="Testing Data")
pred1.plot(label="Prediction with only auto regression part")
plt.legend(loc='best')
```

[20]: <matplotlib.legend.Legend at 0x7e5c46c04160>

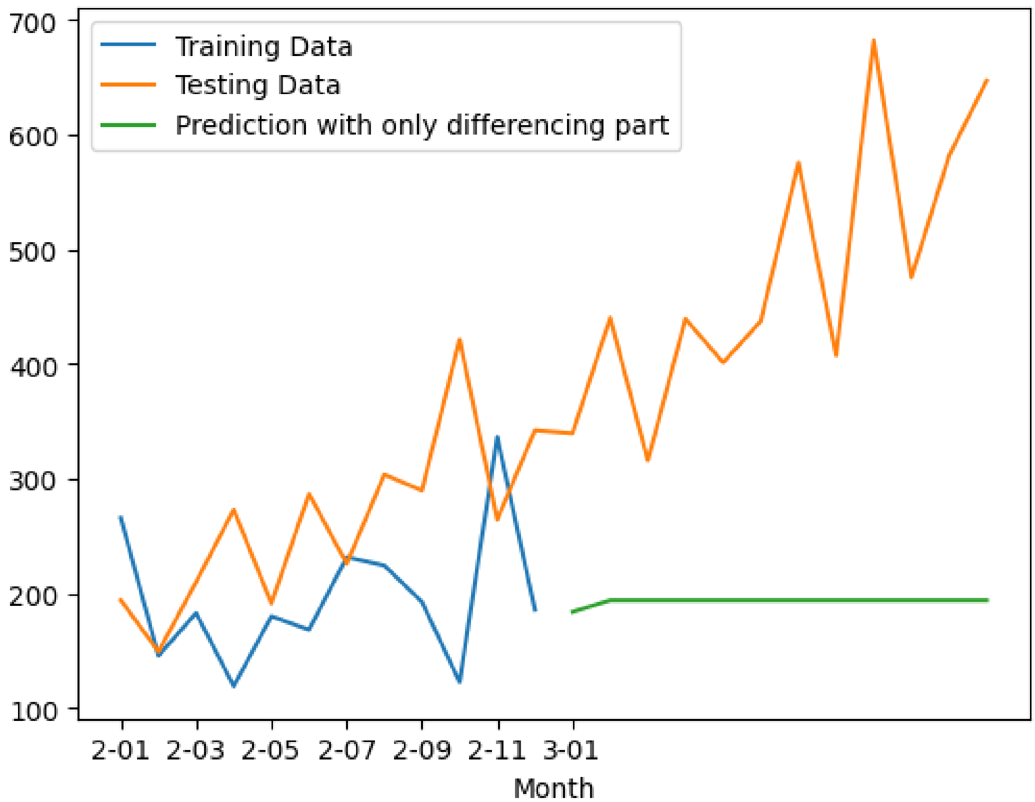
[21]:
```
train['Sales'].plot(label="Training Data")
test['Sales'].plot(label="Testing Data")
pred2.plot(label="Prediction with only differencing part")
plt.legend(loc='best')
```

[21]: <matplotlib.legend.Legend at 0x7e5c48d57820>



[22]:
```
train['Sales'].plot(label="Training Data")
test['Sales'].plot(label="Testing Data")
pred3.plot(label="Prediction with only differencing part")
plt.legend(loc='best')
```

[22]: <matplotlib.legend.Legend at 0x7e5c46a2fca0>



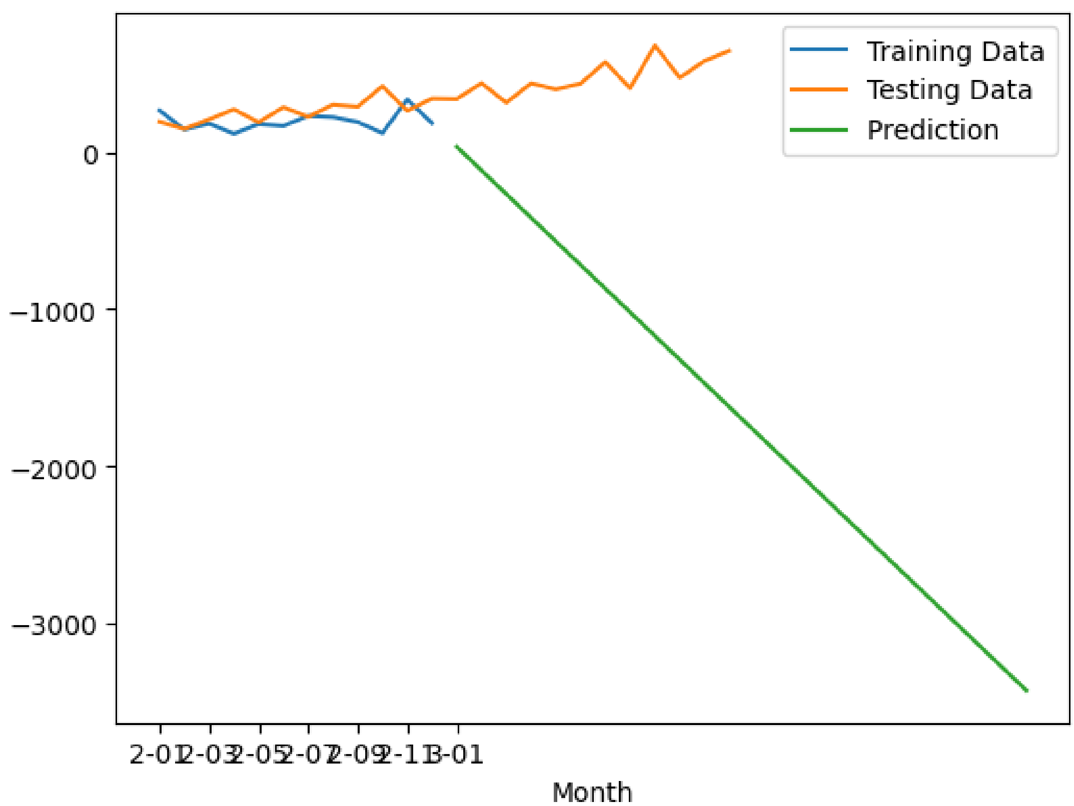
[34]:
```
model_pdq=ARIMA(train,order=(3,2,2)).fit()
pred_pdq=model_pdq.forecast(24)
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
ValueWarning: An unsupported index was provided and will be ignored when e.g.
forecasting.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
ValueWarning: An unsupported index was provided and will be ignored when e.g.
forecasting.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
ValueWarning: An unsupported index was provided and will be ignored when e.g.
forecasting.
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-
packages/statsmodels/tsa/statespace/sarimax.py:966: UserWarning: Non-stationary
starting autoregressive parameters found. Using zeros as starting parameters.
  warn('Non-stationary starting autoregressive parameters'
/usr/local/lib/python3.10/dist-
packages/statsmodels/tsa/statespace/sarimax.py:978: UserWarning: Non-invertible
```

```
starting MA parameters found. Using zeros as starting parameters.
  warn('Non-invertible starting MA parameters found.'
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:836:
ValueWarning: No supported index is available. Prediction results will be given
with an integer index beginning at `start`.
  return get_prediction_index(
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:836:
FutureWarning: No supported index is available. In the next version, calling
this method in a model without a supported index will result in an exception.
  return get_prediction_index(
```

[35]:
```
train['Sales'].plot(label="Training Data")
test['Sales'].plot(label="Testing Data")
pred_pdq.plot(label="Prediction ")
plt.legend(loc='best')
```

[35]: <matplotlib.legend.Legend at 0x7e5c45b02e00>



[36]:
```
print(model_pdq.summary())
```

```
                               SARIMAX Results
==============================================================================
```

```
Dep. Variable:                  Sales   No. Observations:                   12
Model:                 ARIMA(3, 2, 2)   Log Likelihood           -103189353.154
Date:                Fri, 08 Dec 2023   AIC                       206378718.309
Time:                        05:33:59   BIC                       206378720.124
Sample:                             0   HQIC                      206378716.317
                                 - 12
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1      -6.491e-07   2.45e-08    -26.539      0.000   -6.97e-07   -6.01e-07
ar.L2       2.607e-07   3.46e-08      7.544      0.000    1.93e-07    3.28e-07
ar.L3       1.426e-08   7.16e-08      0.199      0.842   -1.26e-07    1.55e-07
ma.L1      -6.491e-07   2.45e-08    -26.539      0.000   -6.97e-07   -6.01e-07
ma.L2       2.607e-07   3.46e-08      7.544      0.000    1.93e-07    3.28e-07
sigma2         0.0014    7.9e-11   1.73e+07      0.000       0.001       0.001
================================================================================
===
Ljung-Box (L1) (Q):                   4.27   Jarque-Bera (JB):
0.26
Prob(Q):                              0.04   Prob(JB):
0.88
Heteroskedasticity (H):               4.25   Skew:
-0.39
Prob(H) (two-sided):                  0.27   Kurtosis:
3.14
================================================================================
===

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-
step).
[2] Covariance matrix is singular or near-singular, with condition number
9.7e+16. Standard errors may be unstable.
```

[37]:

```python
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(test['Sales'],pred_pdq)
mse
```

[37]: 5648812.082357357