00-Introduction-to-Forecasting-Revised

October 26, 2021

[6]:
```python
import pandas as pd
import numpy as np
from IPython.display import Image
```

[13]:
```python
df= pd.read_csv("../data/airline_passengers.
↪csv",index_col="Month",parse_dates=True)
df.index.freq="MS"
df
```

[13]:
```
            Thousands of Passengers
Month
1949-01-01                      112
1949-02-01                      118
1949-03-01                      132
1949-04-01                      129
1949-05-01                      121
…                                 …
1960-08-01                      606
1960-09-01                      508
1960-10-01                      461
1960-11-01                      390
1960-12-01                      432

[144 rows x 1 columns]
```

[14]:
```python
'''
It Goes up 1960, so it means past the year 1960, basically entering 1961.
That is the future.

And according to this data set that we don't have data for.
So later on, towards the very end, we're going to forecast into the early 60s.

So we'll try to forecast maybe a one to three years ahead and see what we␣
↪predict as far as the thousands
of passengers flying for every month, three years into the future.
'''
df1.tail()
```

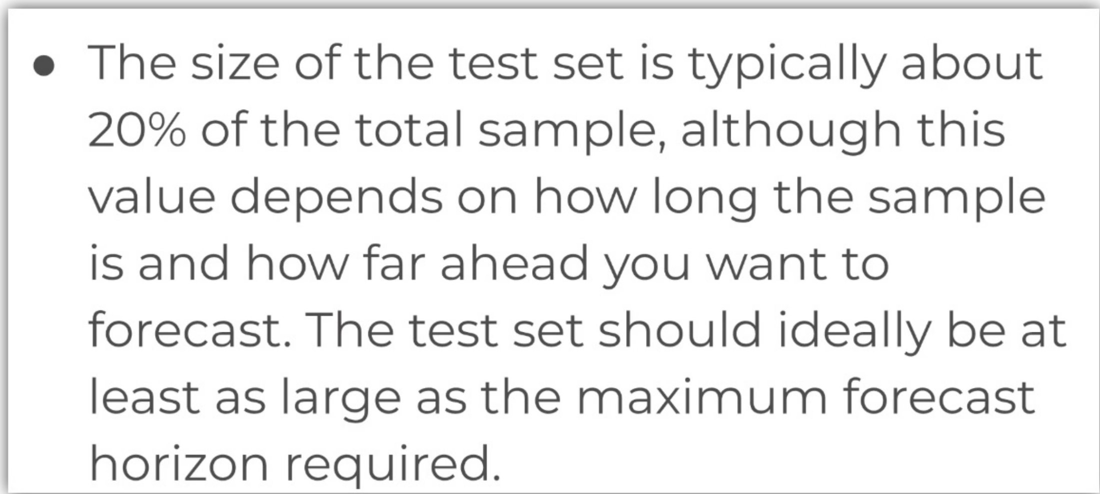
[14]:
```
            Thousands of Passengers
Month
1960-08-01                      606
1960-09-01                      508
1960-10-01                      461
1960-11-01                      390
1960-12-01                      432
```

[33]:
```
Image('/Users/subhasish/Documents/APPLE/SUBHASISH/Development/GIT/Interstellar/
 ↪SB-AI-DEV/ML/SB/TimeSeries/Jose Portilla/Python for Time Series Data␣
 ↪Analysis/Image/2021-10-26_10-23-43.jpg')

'''
Test sets in Time series will be the most recent end of the data.

So if we were to lay out our time series flat with time going forward or to the␣
 ↪right, we'd have the
first portion, the larger portion be the training data, and then we would have␣
 ↪the most recent and
be our test data.

So we will fit our model on the training data and then forecast off the␣
 ↪training data to the same length
of time that our test data is and then compare our forecasted results to the␣
 ↪real test data that we
already know the correct answers for.

But a really common question is how do we decide how large that portion of the␣
 ↪data should be?
The test data?
And there's no 100 percent correct answer here, but typically the size of the␣
 ↪test is about 20 percent
of the total sample, and this really depends on how long the sample is and how␣
 ↪far ahead you want to
forecast.

What you should really keep in mind is instead of this 80, 20 percent split, is␣
 ↪that the test size
should ideally be at least as large as the maximum forecast horizon required.

So what that means is if you intend to predict one year into the future, then␣
 ↪your test data should
be at least one year in length.

Keep in mind, however, the longer the forecast horizon, the more likely your␣
 ↪prediction will become
```

```
less accurate just because you're starting to predict more and more and there's␣
↪more noise added in
and now you're predicting off a prediction.
'''
print()
```

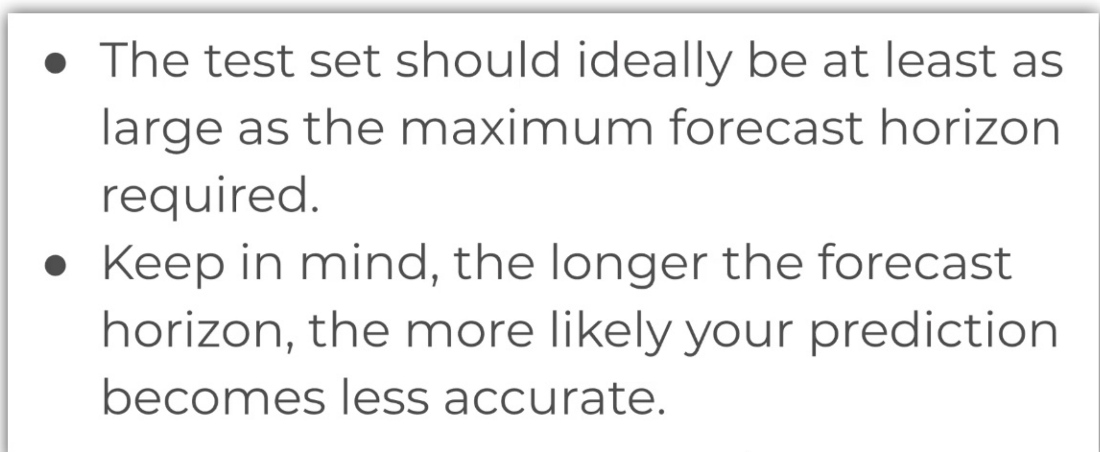
[16]:
```
Image("/Users/subhasish/Documents/APPLE/SUBHASISH/Development/GIT/Interstellar/
↪SB-AI-DEV/ML/SB/TimeSeries/Jose Portilla/Python for Time Series Data␣
↪Analysis/Image/2021-10-26_15-39-36.jpg")
```

[16]:



[17]:
```
Image("/Users/subhasish/Documents/APPLE/SUBHASISH/Development/GIT/Interstellar/
↪SB-AI-DEV/ML/SB/TimeSeries/Jose Portilla/Python for Time Series Data␣
↪Analysis/Image/2021-10-26_15-42-04.jpg")
```

[17]:



[20]:
```
'''
Let's go ahead and perform the train to split and fortunately, the train to␣
 ↪split is essentially just
an indexing command and you can either do it by the timestamp or by the index␣
 ↪for the integer location

We simply say grab our entire data frame, which is just here, essentially a␣
 ↪single column, and then
say df.iLoc And then go : all the way from the beginning, up to some index␣
 ↪position 109
'''
df.info()
train_data =df.iloc[:119]
test_data=df.iloc[118:]
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 144 entries, 1949-01-01 to 1960-12-01
Freq: MS
Data columns (total 1 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Thousands of Passengers  144 non-null    int64
dtypes: int64(1)
memory usage: 2.2 KB
```

[22]:
```
'''
Now it's time to fit the model to the training data.

It will say exponential smoothing, grab our training data and the column from␣
 ↪the training data is
just want a single series is thousands of passengers.

because we're going to do exponential smoothing whether we want a
multiplicative trend or a seasonal trend.

I'll go ahead and use a multiplicative trend and a multiplicative seasonal␣
 ↪component.
So we're going to have a couple of different parameters here.trend='mul',␣
 ↪seasonal='mul'

because 12 entries per seasonal period, 12 months per year so␣
 ↪seasonal_periods=12
'''
from statsmodels.tsa.holtwinters import ExponentialSmoothing
```

```python
fitted_model=ExponentialSmoothing(train_data["Thousands of Passengers"],
                                  trend='mul',
                                  seasonal='mul',
                                  seasonal_periods=12).fit()
```

```
/Users/subhasish/opt/anaconda3/envs/ML/lib/python3.8/site-
packages/statsmodels/tsa/holtwinters/model.py:80: RuntimeWarning: overflow
encountered in matmul
  return err.T @ err
```

[23]:
```python
'''
And now it's time to forecast on the test data and then compare it to the test␣
 ↪data.

off this fitted model object.
You should be able to call that forecast and then it's up to you to provide how␣
 ↪many periods you want
to forecast into the future.

Now, every row is essentially one month of information.
So that means if I wanted to forecast one year into the future, I would do 12␣
 ↪periods or if I wanted
to do three years into the future, I would do 36 periods because 12 * 3=36.
'''

test_predictions=fitted_model.forecast(36)
```

[24]:
```python
'''
So if we take a look at what test predictions is, we can see here it's␣
 ↪essentially a series where we're
predicting a certain value for a date.
'''
test_predictions
```

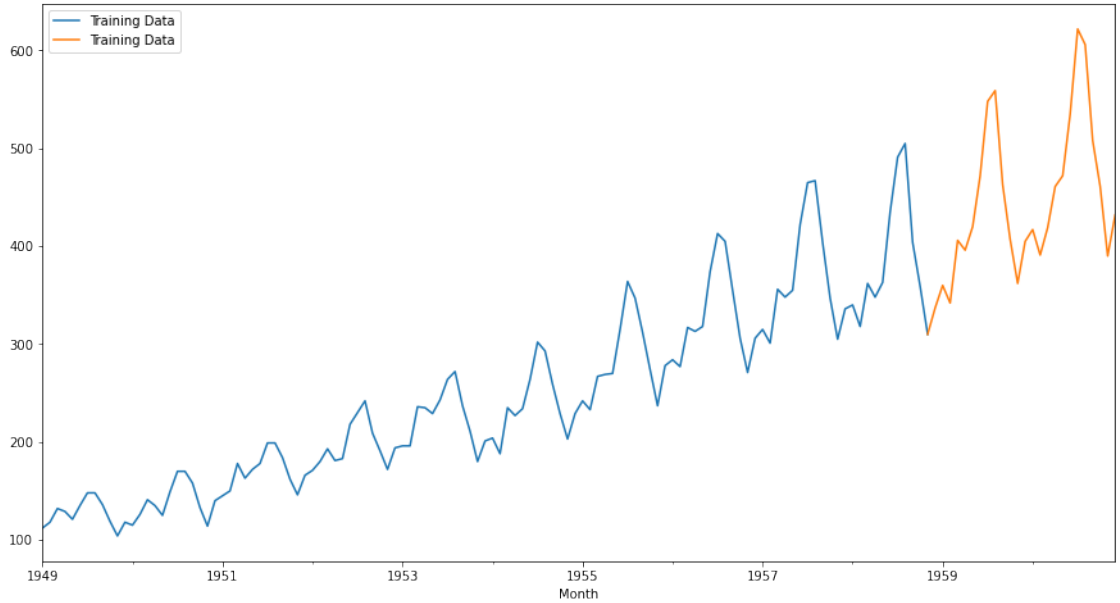
[24]:
```
1958-12-01    349.325653
1959-01-01    359.117285
1959-02-01    342.850919
1959-03-01    400.231275
1959-04-01    394.052822
1959-05-01    413.967619
1959-06-01    496.062375
1959-07-01    554.592778
1959-08-01    558.745452
1959-09-01    453.981065
1959-10-01    401.911452
1959-11-01    352.025352
1959-12-01    393.246091
```

```
1960-01-01    404.268817
1960-02-01    385.957294
1960-03-01    450.552037
1960-04-01    443.596772
1960-05-01    466.015441
1960-06-01    558.431905
1960-07-01    624.321289
1960-08-01    628.996077
1960-09-01    511.059746
1960-10-01    452.443462
1960-11-01    396.285221
1960-12-01    442.688611
1961-01-01    455.097216
1961-02-01    434.483400
1961-03-01    507.199589
1961-04-01    499.369844
1961-05-01    524.607194
1961-06-01    628.643107
1961-07-01    702.816712
1961-08-01    708.079256
1961-09-01    575.314884
1961-10-01    509.328820
1961-11-01    446.109849
Freq: MS, dtype: float64
```

[29]:

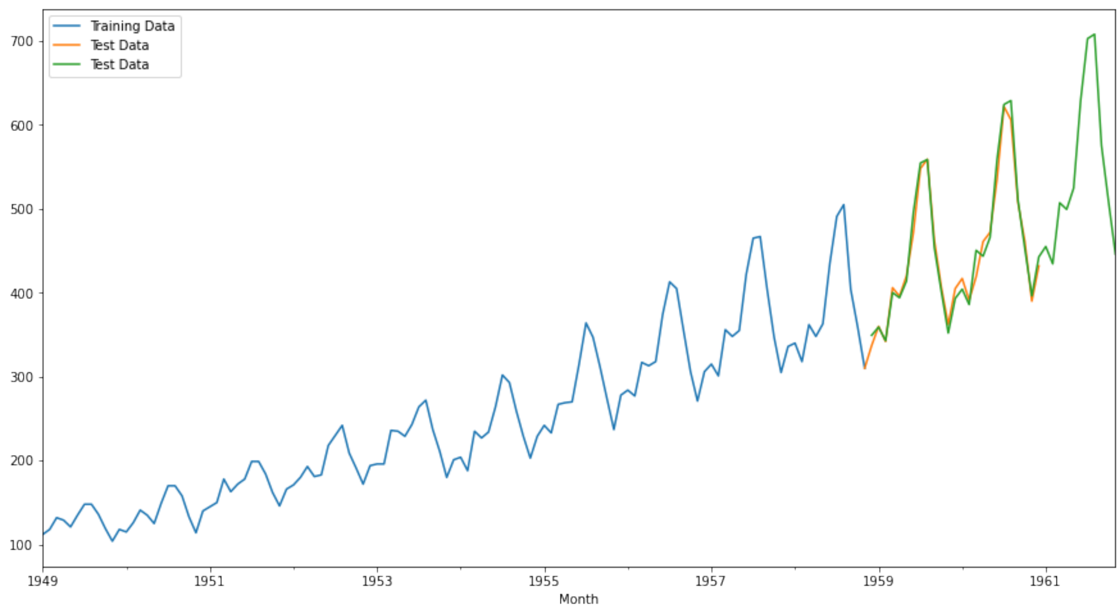
```
'''
So we're going to do now is plot this against our real data.
So the first plot, the training in the test data, then we'll plot our␣
↪predictions
'''

train_data["Thousands of Passengers"].plot(legend=True,label="Training Data",␣
↪figsize=(15,8))
test_data["Thousands of Passengers"].plot(legend=True,label="Test Data",␣
↪figsize=(15,8))
```

[29]: <AxesSubplot:xlabel='Month'>




```
[31]: '''
Now, what I want to do is I want to see how well did my predictions actually␣
↪perform

We have our original training data and then we can see the test in orange and␣
↪the prediction in green.

Our prediction seems to be more or less on top of our test data.
'''
train_data["Thousands of Passengers"].plot(legend=True,label="Training Data",␣
↪figsize=(15,8))
test_data["Thousands of Passengers"].plot(legend=True,label="Test Data",␣
↪figsize=(15,8))
test_predictions.plot(legend=True,label="Test Data", figsize=(15,8))
```

```
[31]: <AxesSubplot:xlabel='Month'>
```

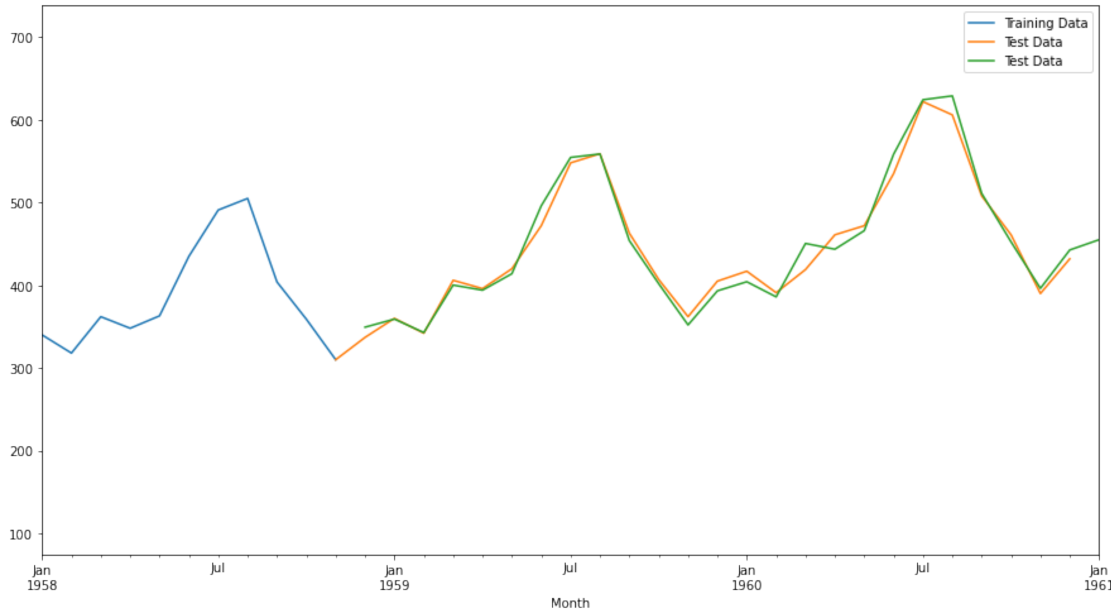


[32]:

```
'''
And in fact, we can zoom in on this to see what's going on in more detail.

So just off one of these the last one we can say␣
↪xlim=['1958-01-01','1961-01-01'] and let's go ahead and set the X limits to␣
↪what
we the range we were predicting for, which was essentially the beginning of␣
↪1958,to 1961

And you can see that we're definitely picking up a lot of the information.

We're able to pick up that seasonality.
But in some cases, our prediction is maybe lagging a little bit or it's under␣
↪predicting the results
and sometimes it's overprotecting kind of on the downturn's.

We can see visually here that we're performing pretty well, but how do we␣
↪actually quantify this?
So we need to learn about a couple of the evaluation metrics so we can quantify␣
↪just how off our prediction
is from our test data.

'''

train_data["Thousands of Passengers"].plot(legend=True,label="Training Data",␣
↪figsize=(15,8))
```

```
test_data["Thousands of Passengers"].plot(legend=True,label="Test Data",␣
↪figsize=(15,8))
test_predictions.plot(legend=True,label="Test Data",␣
↪figsize=(15,8),xlim=['1958-01-01','1961-01-01'])
```

[32]: <AxesSubplot:xlabel='Month'>



[ ]: