forecasting_online_retail

May 31, 2021

The final part of the project would be to provide a forecast of the weekly agrregated sales/revenue, for 4 weeks in advance. Before we proceed, the necessary Python libraries are loaded along with the final data set from the first part of the project.

[1]:
```
# importing necessary Python libraries

import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
from matplotlib import pylab
from pylab import *
from pylab import rcParams

import seaborn as sns
import warnings
import itertools
warnings.filterwarnings("ignore")
plt.style.use("fivethirtyeight")

import statsmodels.api as sm
import matplotlib
%matplotlib inline
matplotlib.rcParams["axes.labelsize"] = 14
matplotlib.rcParams["xtick.labelsize"] = 12
matplotlib.rcParams["ytick.labelsize"] = 12
matplotlib.rcParams["text.color"] = "k"

import sklearn as sk
from sklearn.metrics import r2_score,mean_squared_error, mean_absolute_error
from sklearn import preprocessing

import fbprophet
from fbprophet import Prophet
from fbprophet.plot import add_changepoints_to_plot
from fbprophet.diagnostics import cross_validation
from fbprophet.diagnostics import performance_metrics
```

```python
from fbprophet.plot import plot_cross_validation_metric
```

[2]:
```python
# reload the dataset from part 1
data = pd.read_csv("data.csv").drop(['Unnamed: 0'],axis=1)
data.head()
```

[2]:
```
   Invoice StockCode                          Description  Quantity  \
0   489434     85048  15CM CHRISTMAS GLASS BALL 20 LIGHTS        12
1   489434    79323P                   PINK CHERRY LIGHTS        12
2   489434    79323W                  WHITE CHERRY LIGHTS        12
3   489434     22041         RECORD FRAME 7" SINGLE SIZE         48
4   489434     21232       STRAWBERRY CERAMIC TRINKET BOX        24

           InvoiceDate  Price  CustomerID         Country  Revenue Year-Week
0  2009-12-01 07:45:00   6.95       13085  United Kingdom     83.4   2009-49
1  2009-12-01 07:45:00   6.75       13085  United Kingdom     81.0   2009-49
2  2009-12-01 07:45:00   6.75       13085  United Kingdom     81.0   2009-49
3  2009-12-01 07:45:00   2.10       13085  United Kingdom    100.8   2009-49
4  2009-12-01 07:45:00   1.25       13085  United Kingdom     30.0   2009-49
```

[3]:
```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 775758 entries, 0 to 775757
Data columns (total 10 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   Invoice      775758 non-null  int64
 1   StockCode    775758 non-null  object
 2   Description  775758 non-null  object
 3   Quantity     775758 non-null  int64
 4   InvoiceDate  775758 non-null  object
 5   Price        775758 non-null  float64
 6   CustomerID   775758 non-null  int64
 7   Country      775758 non-null  object
 8   Revenue      775758 non-null  float64
 9   Year-Week    775758 non-null  object
dtypes: float64(2), int64(3), object(5)
memory usage: 59.2+ MB
```

Having a look at the data information, it is noticed that “InvoiceDate” is an ‘object’ type.

Thus, “InvoiceDate” is again converted to ‘datetime’ type.

[4]:
```python
data["InvoiceDate"] = pd.to_datetime(data["InvoiceDate"])
```

Focusing on the goal of the task and taking into account that Prophet is going to be used for forecasting, “InvoiceDate” and “Revenue” are extracted from the final set. A new data frame "weekly_data" is generated including these two features. Since we the aim is to forecast weekly sales, the data should be grouped by week.

[5]:
```
weekly_data = data[["InvoiceDate","Revenue"]]
weekly_data.rename(columns={"InvoiceDate": "ds", "Revenue": "y"},inplace=True)
weekly_data.set_index("ds")
print(weekly_data)
print(weekly_data.info())
```

```
                        ds       y
0      2009-12-01 07:45:00   83.40
1      2009-12-01 07:45:00   81.00
2      2009-12-01 07:45:00   81.00
3      2009-12-01 07:45:00  100.80
4      2009-12-01 07:45:00   30.00
...                    ...     ...
775753 2011-12-09 12:50:00   10.20
775754 2011-12-09 12:50:00   12.60
775755 2011-12-09 12:50:00   16.60
775756 2011-12-09 12:50:00   16.60
775757 2011-12-09 12:50:00   14.85

[775758 rows x 2 columns]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 775758 entries, 0 to 775757
Data columns (total 2 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   ds      775758 non-null  datetime64[ns]
 1   y       775758 non-null  float64
dtypes: datetime64[ns](1), float64(1)
memory usage: 11.8 MB
None
```

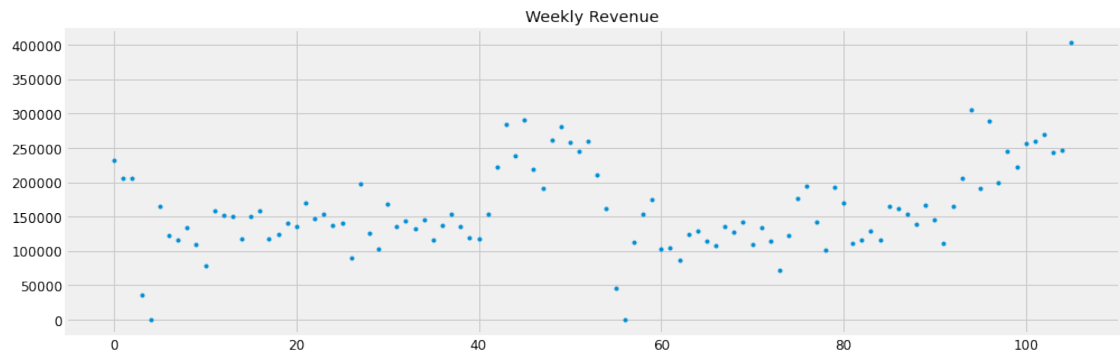
[6]:
```
##### AGGREGATE DAILY TO WEEKLY #######
weekly_data = weekly_data.set_index("ds").resample("W").sum()
weekly_data.reset_index(inplace=True)
weekly_data
```

[6]:
```
            ds          y
0   2009-12-06  231127.70
1   2009-12-13  205548.06
2   2009-12-20  206004.32
3   2009-12-27   35327.44
4   2010-01-03       0.00
..         ...        ...
101 2011-11-13  259098.75
102 2011-11-20  269021.49
```

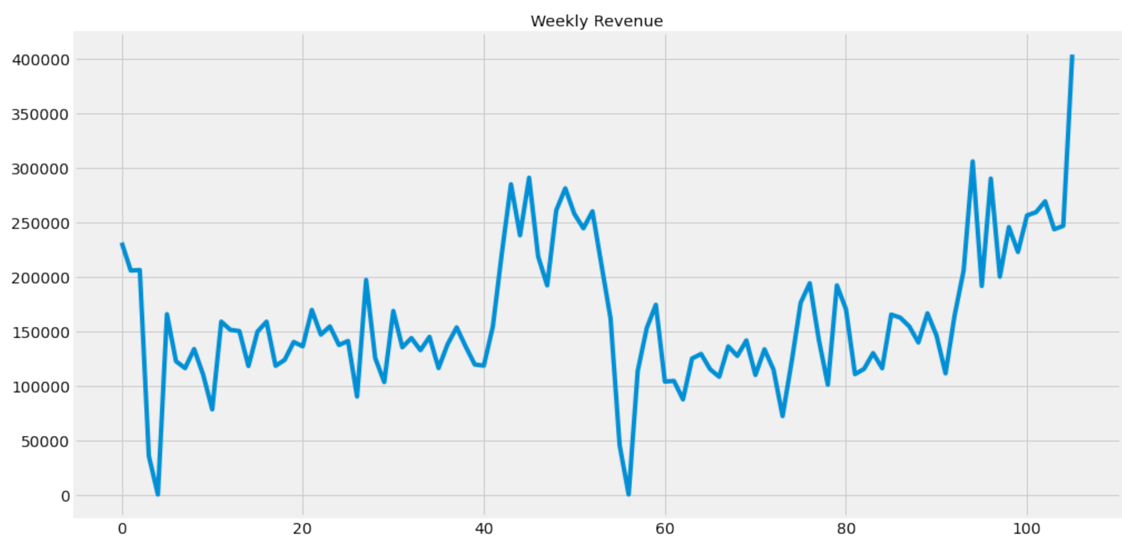
```
103 2011-11-27  243407.13
104 2011-12-04  246538.76
105 2011-12-11  403741.12

[106 rows x 2 columns]
```

The dataframe 'weekly_data' is prepared this way to be fitted in the Prophet() model that has specific requirements for the characteristics of the inputs, such as the columns to be named to 'ds' ("InvoiceDate") and 'y' ("Revenue"). After all, a scatter plot and a time series plot of the Weekly Revenue are indicated.

[7]:
```
weekly_data["y"].plot(style='.', figsize=(15,5), title='Weekly Revenue')
plt.show()
```



[8]:
```
weekly_data.y.plot(figsize=(15,8), title= 'Weekly Revenue', fontsize=14)
plt.show()
```



Not much can be seen from the two plots, although it is interesting that there are 0 points in the plots following a similar pattern. The next step here would then be to split the data set into training and testing sets. Since, there are not many observations for the weekly aggregated sales, a 50%-50% split is made.

[9]:
```
# splitting dataframe by row index
train_weekly_data = weekly_data.iloc[:53,:]
test_weekly_data = weekly_data.iloc[53:,:]
print("Shape of new dataframes - {} , {}".format(train_weekly_data.shape,␣
↪test_weekly_data.shape))
```

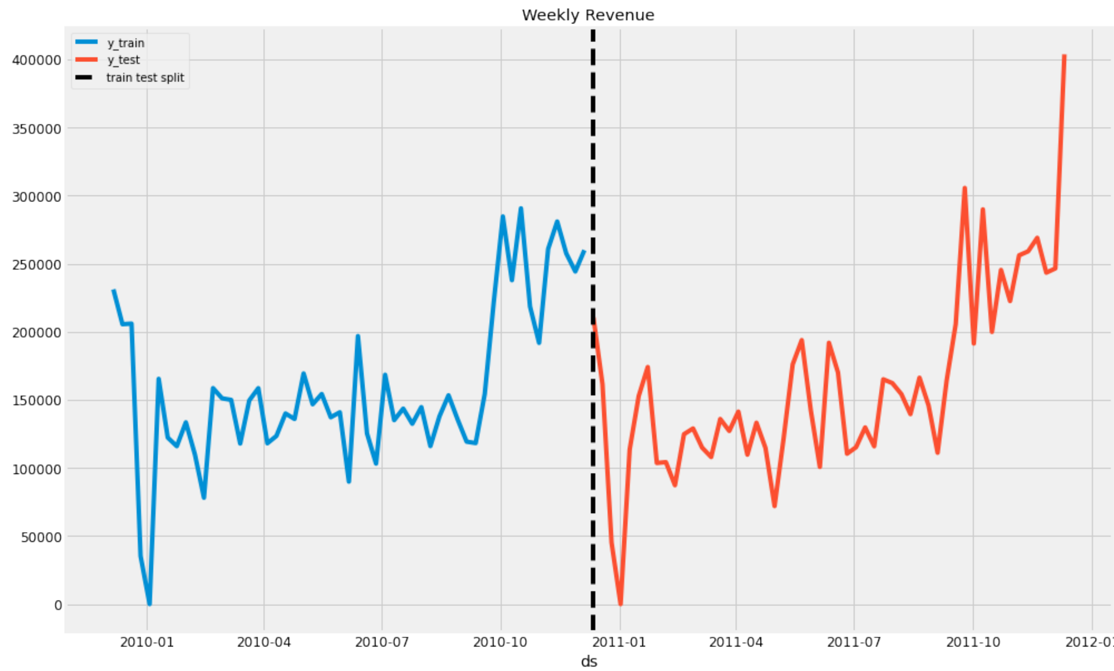
```
Shape of new dataframes - (53, 2) , (53, 2)
```

[10]:
```
print(train_weekly_data.head())
print(test_weekly_data.tail())
```

```
          ds          y
0 2009-12-06  231127.70
1 2009-12-13  205548.06
2 2009-12-20  206004.32
3 2009-12-27   35327.44
4 2010-01-03       0.00
            ds          y
101 2011-11-13  259098.75
102 2011-11-20  269021.49
103 2011-11-27  243407.13
104 2011-12-04  246538.76
105 2011-12-11  403741.12
```

Before the Prophet model is applied to the data, a visualization of the training and testing data sets along with their split is illustrated in the following plot.

[11]:
```
# Define threshold date.
threshold_date = pd.to_datetime('2010-12-12')

rcParams['figure.figsize'] = 15, 10

fig, ax = plt.subplots()
sns.lineplot(x='ds', y='y', label='y_train', data=train_weekly_data, ax=ax)
sns.lineplot(x='ds', y='y', label='y_test', data=test_weekly_data, ax=ax)
ax.axvline(threshold_date, color='black', linestyle='--', label='train test␣
↪split')
ax.legend(loc='upper left')
ax.set(title='Weekly Revenue', ylabel='');
```



In a nutshell, Prophet is based on a generalized additive model (GAM) where non-linear trends are fit with yearly, weekly, and daily seasonality, plus holiday effects. It seems that Prophet requires several seasons of historical data, although in this case there is a limited amount of observed of weekly sales. Nevertheless, since there is no past experience with Prophet, it is of interest to further investigate it.

[12]:
```
# check prophet version
import fbprophet
# print version number
print('Prophet %s' % fbprophet.__version__)
```

```
Prophet 0.7.1
```

In the first implementation of the model, none of the parameters is specified and it is fitted on the training set with the default values. Then, predictions are made for the tesitng set.

[13]:
```
##### 1. Naive Approach -- Default parameter values
```

[14]:
```
# call prophet model
model = Prophet()
# fit the model to the training set
model.fit(train_weekly_data)
# predict on testing period
future = model.make_future_dataframe(periods=test_weekly_data.shape[0], freq='W')
# generate predictions
forecast = model.predict(future)
```

```
forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail()
```

```
INFO:fbprophet:Disabling yearly seasonality. Run prophet with
yearly_seasonality=True to override this.
INFO:fbprophet:Disabling weekly seasonality. Run prophet with
weekly_seasonality=True to override this.
INFO:fbprophet:Disabling daily seasonality. Run prophet with
daily_seasonality=True to override this.
```

[14]:

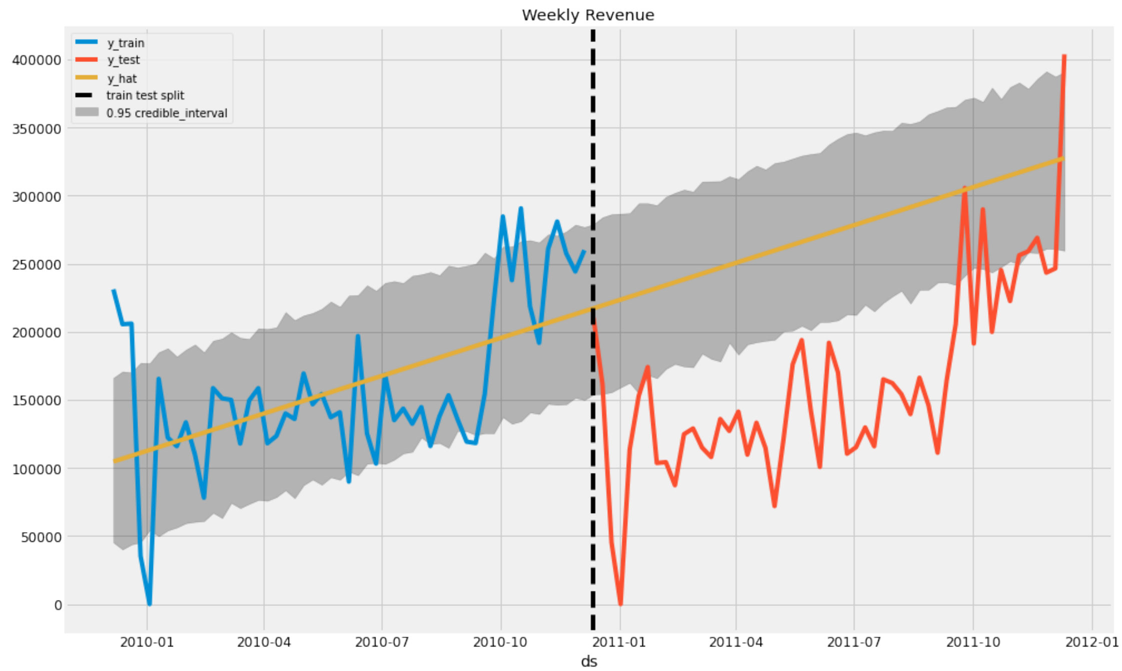
	ds	yhat	yhat_lower	yhat_upper
101	2011-11-13	319115.091712	258353.834031	378487.849184
102	2011-11-20	321237.983850	257942.286225	385697.321051
103	2011-11-27	323360.875988	261054.608962	391247.534573
104	2011-12-04	325483.768126	260989.954713	387423.028346
105	2011-12-11	327606.660265	259616.900542	390791.192510

[15]:

```
# split the predictions into training and testing
mask2 = forecast['ds'] < threshold_date

forecast_train = forecast[mask2]
forecast_test = forecast[~ mask2]
```

[16]:

```
# plot Weekly Revenue for the training and testing sets along with model's␣
 ↪predictive values
fig, ax = plt.subplots()

from pylab import rcParams
rcParams['figure.figsize'] = 20, 10

ax.fill_between(
    x=forecast['ds'],
    y1=forecast['yhat_lower'],
    y2=forecast['yhat_upper'],
    color='black',
    alpha = 0.25,
    label=r'0.95 credible_interval'
)

sns.lineplot(x='ds', y='y', label='y_train', data=train_weekly_data, ax=ax)
sns.lineplot(x='ds', y='y', label='y_test', data=test_weekly_data, ax=ax)
sns.lineplot(x='ds', y='yhat', label='y_hat', data=forecast, ax=ax)
ax.axvline(threshold_date, color='black', linestyle='--', label='train test␣
 ↪split')
ax.legend(loc='upper left')
ax.set(title='Weekly Revenue', ylabel='');
```



In the end , along with the dates, the predicted values 'yhat' are obtained as well as a confidence interval ['yhat_lower','yhat_upper']. It can be seen that the predictions are an increasing linear function that doesn't capture any trend or flactuantions in the data. In addition, the R^2 value and the Mean Absolute Error for both training and testing are obtained. The default version of prophet performs really bad for this data set.

[17]:
```
print('R2 Train: {}'.format(r2_score(y_true=train_weekly_data['y'],␣
 ↪y_pred=forecast_train['yhat'])))
print('R2 Test: {}'.format(r2_score(y_true=test_weekly_data['y'],␣
 ↪y_pred=forecast_test['yhat'])))
print('---'*10)
print('MAE Train: {}'.format(mean_absolute_error(y_true=train_weekly_data['y'],␣
 ↪y_pred=forecast_train['yhat'])))
print('MAE Test: {}'.format(mean_absolute_error(y_true=test_weekly_data['y'],␣
 ↪y_pred=forecast_test['yhat'])))
```

```
R2 Train: 0.2971092986036904
R2 Test: -2.11595971661552
------------------------------
MAE Train: 39902.67553020685
MAE Test: 113467.9774054433
```

The prophet allows the specification of various parameters, so another prophet model will be fitted by setting "seasonality_mode" : multiplicative, "daily_seasonality" : True, "weekly_seasonality" : True and "yearly_seasonality" : True. The same process is followed in order to gain some insight and draw conclusions.

[18]:
```
##### 2. Approach - Include daily, weekly and yearly seasonality and mode of␣
↪seasonality.
```

[19]:
```
model_2 = Prophet(seasonality_mode = 'multiplicative', daily_seasonality = True,␣
↪weekly_seasonality = True, yearly_seasonality = True)
```

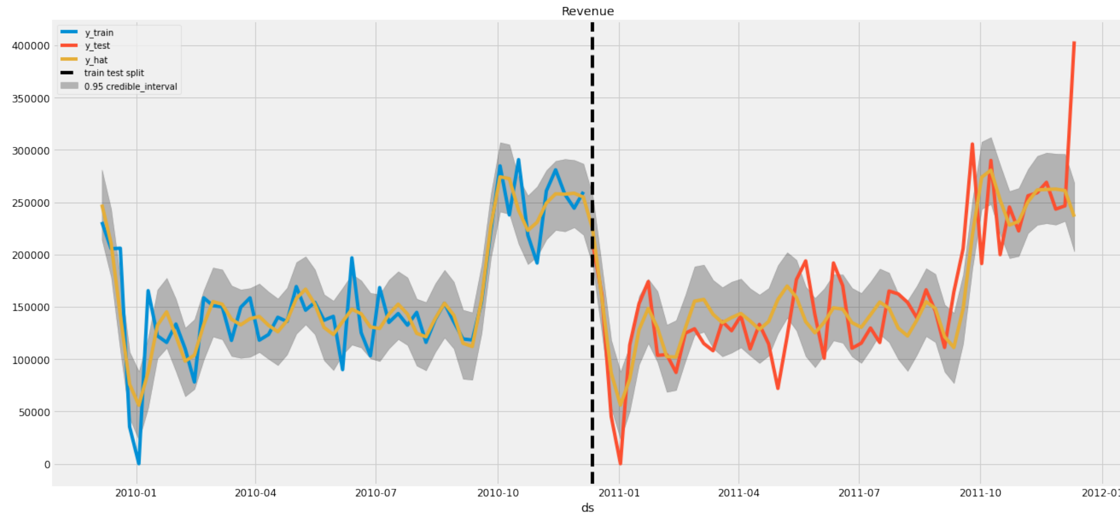
[20]:
```
model_2.fit(train_weekly_data)
```

[20]: <fbprophet.forecaster.Prophet at 0x120e780a0>

[21]:
```
# Extend dates and features.
future_2 = model_2.make_future_dataframe(periods=test_weekly_data.shape[0],␣
↪freq='W')
```

[22]:
```
# Generate predictions.
forecast_2 = model_2.predict(future_2)
```

[23]:
```
fig, ax = plt.subplots()

from pylab import rcParams
rcParams['figure.figsize'] = 5, 10

ax.fill_between(
    x=forecast_2['ds'],
    y1=forecast_2['yhat_lower'],
    y2=forecast_2['yhat_upper'],
    color='black',
    alpha = 0.25,
    label=r'0.95 credible_interval'
)

sns.lineplot(x='ds', y='y', label='y_train', data=train_weekly_data, ax=ax)
sns.lineplot(x='ds', y='y', label='y_test', data=test_weekly_data, ax=ax)
sns.lineplot(x='ds', y='yhat', label='y_hat', data=forecast_2, ax=ax)
ax.axvline(threshold_date, color='black', linestyle='--', label='train test␣
↪split')
ax.legend(loc='upper left')
ax.set(title='Revenue', ylabel='');
```



[24]:
```
mask2 = forecast_2['ds'] < threshold_date

forecast_train_2 = forecast_2[mask2]
forecast_test_2 = forecast_2[~ mask2]
```

[25]:
```
print('R2 Train: {}'.format(r2_score(y_true=train_weekly_data['y'],␣
 ↪y_pred=forecast_train_2['yhat'])))
print('R2 Test: {}'.format(r2_score(y_true=test_weekly_data['y'],␣
 ↪y_pred=forecast_test_2['yhat'])))
print('---'*10)
print('MAE Train: {}'.format(mean_absolute_error(y_true=train_weekly_data['y'],␣
 ↪y_pred=forecast_train_2['yhat'])))
print('MAE Test: {}'.format(mean_absolute_error(y_true=test_weekly_data['y'],␣
 ↪y_pred=forecast_test_2['yhat'])))
```

```
R2 Train: 0.8167461013856192
R2 Test: 0.6539555752827162
------------------------------
MAE Train: 18666.73275082345
MAE Test: 28783.227668495907
```

The second model, as it can be seen in the figure above, has a good fit to the data. This can also be seen in the improved R^2 score (=0.65) and MAE (=28783), meaning that 65% of the variance is explained in the model and that on average, the forecast's distance from the true weekly revenue is 28783.

It is clear that the performance of the model has been improved by taking into account some of the parameters of the model. After seeing this, it would therefore be a good idea to also include a holiday effect in the model. The same process is followed.

[26]:
```
##### 3rd Approach - Include Holidays
```

[27]:
```
model_3 = Prophet(seasonality_mode = 'multiplicative', daily_seasonality = True,␣
    ↪weekly_seasonality = True, yearly_seasonality = True)
model_3.add_country_holidays(country_name='UK')
model_3.fit(train_weekly_data)
# List the holiday names
model_3.train_holiday_names
future_3 = model_3.make_future_dataframe(periods=test_weekly_data.shape[0],␣
    ↪freq='W')
# Generate predictions.
forecast_3 = model_3.predict(future_3)
```

[28]:
```
print(model_3.train_holiday_names)
forecast_3
```

```
0                                          New Year's Day
1                               New Year Holiday [Scotland]
2                          St. Patrick's Day [Northern Ireland]
3                        Battle of the Boyne [Northern Ireland]
4                             Summer Bank Holiday [Scotland]
5                               St. Andrew's Day [Scotland]
6                                           Christmas Day
7                                             Good Friday
8             Easter Monday [England/Wales/Northern Ireland]
9                                                 May Day
10                                    Spring Bank Holiday
11     Late Summer Bank Holiday [England/Wales/Northe...
12                                             Boxing Day
13                                  Boxing Day (Observed)
14                    New Year Holiday [Scotland] (Observed)
15                               Christmas Day (Observed)
dtype: object
```

[28]:
```
             ds          trend     yhat_lower     yhat_upper    trend_lower  \
0    2009-12-06  156224.046112  212789.923529  280337.321948  156224.046112
1    2009-12-13  156274.741464  178450.364679  243134.051995  156274.741464
2    2009-12-20  156325.436816  108143.811697  175482.500009  156325.436816
3    2009-12-27  156376.132168   43955.813005  109625.178362  156376.132168
4    2010-01-03  156426.827520   21443.600823   90308.821602  156426.827520
..          ...            ...            ...            ...            ...
101  2011-11-13  161344.276611  226120.613543  293892.767858  161344.261900
102  2011-11-20  161394.971963  227730.907219  294348.715731  161394.956822
103  2011-11-27  161445.667314  230576.412021  299627.846963  161445.651725
104  2011-12-04  161496.362665  227817.977968  293565.392295  161496.346555
105  2011-12-11  161547.058017  204486.863285  272191.595112  161547.041386

       trend_upper  Battle of the Boyne [Northern Ireland]  \
0    156224.046112                                     0.0
```

```
1    156274.741464                                      0.0
2    156325.436816                                      0.0
3    156376.132168                                      0.0
4    156426.827520                                      0.0
..             ...                                      ...
101  161344.289810                                      0.0
102  161394.985546                                      0.0
103  161445.681394                                      0.0
104  161496.377146                                      0.0
105  161547.072802                                      0.0

     Battle of the Boyne [Northern Ireland]_lower  \
0                                             0.0
1                                             0.0
2                                             0.0
3                                             0.0
4                                             0.0
..                                            ...
101                                           0.0
102                                           0.0
103                                           0.0
104                                           0.0
105                                           0.0

     Battle of the Boyne [Northern Ireland]_upper  Boxing Day  ...    weekly  \
0                                             0.0         0.0  ...  0.002189
1                                             0.0         0.0  ...  0.002189
2                                             0.0         0.0  ...  0.002189
3                                             0.0         0.0  ...  0.002189
4                                             0.0         0.0  ...  0.002189
..                                            ...         ...  ...       ...
101                                           0.0         0.0  ...  0.002189
102                                           0.0         0.0  ...  0.002189
103                                           0.0         0.0  ...  0.002189
104                                           0.0         0.0  ...  0.002189
105                                           0.0         0.0  ...  0.002189

     weekly_lower  weekly_upper    yearly  yearly_lower  yearly_upper  \
0        0.002189      0.002189  0.581185      0.581185      0.581185
1        0.002189      0.002189  0.344064      0.344064      0.344064
2        0.002189      0.002189 -0.097372     -0.097372     -0.097372
3        0.002189      0.002189 -0.519522     -0.519522     -0.519522
4        0.002189      0.002189 -0.647794     -0.647794     -0.647794
..            ...           ...       ...           ...           ...
101      0.002189      0.002189  0.617208      0.617208      0.617208
102      0.002189      0.002189  0.619225      0.619225      0.619225
103      0.002189      0.002189  0.621247      0.621247      0.621247
```

```
104       0.002189       0.002189   0.611978       0.611978       0.611978
105       0.002189       0.002189   0.456831       0.456831       0.456831

     additive_terms  additive_terms_lower  additive_terms_upper           yhat
0               0.0                   0.0                   0.0  247817.072187
1               0.0                   0.0                   0.0  210841.481037
2               0.0                   0.0                   0.0  141902.297827
3               0.0                   0.0                   0.0   75934.060197
4               0.0                   0.0                   0.0   55893.480937
..              ...                   ...                   ...            ...
101             0.0                   0.0                   0.0  261751.457236
102             0.0                   0.0                   0.0  262159.246770
103             0.0                   0.0                   0.0  262567.937012
104             0.0                   0.0                   0.0  261153.567444
105             0.0                   0.0                   0.0  236172.002496

[106 rows x 73 columns]
```

[29]:

```
mask2 = forecast_3['ds'] < threshold_date

forecast_train_3 = forecast_3[mask2]
forecast_test_3 = forecast_3[~ mask2]
```

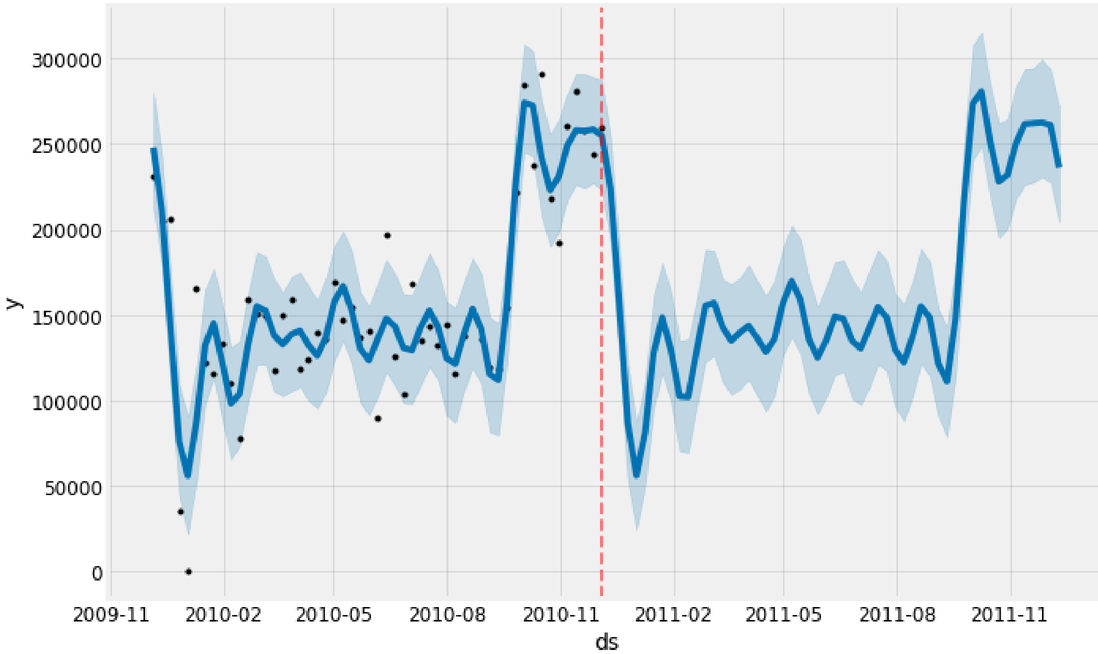
[30]:

```
print('r2 train: {}'.format(r2_score(y_true=train_weekly_data['y'],␣
 ↪y_pred=forecast_train_3['yhat'])))
print('r2 test: {}'.format(r2_score(y_true=test_weekly_data['y'],␣
 ↪y_pred=forecast_test_3['yhat'])))
print('---'*10)
print('mae train: {}'.format(mean_absolute_error(y_true=train_weekly_data['y'],␣
 ↪y_pred=forecast_train_3['yhat'])))
print('mae test: {}'.format(mean_absolute_error(y_true=test_weekly_data['y'],␣
 ↪y_pred=forecast_test_3['yhat'])))
```

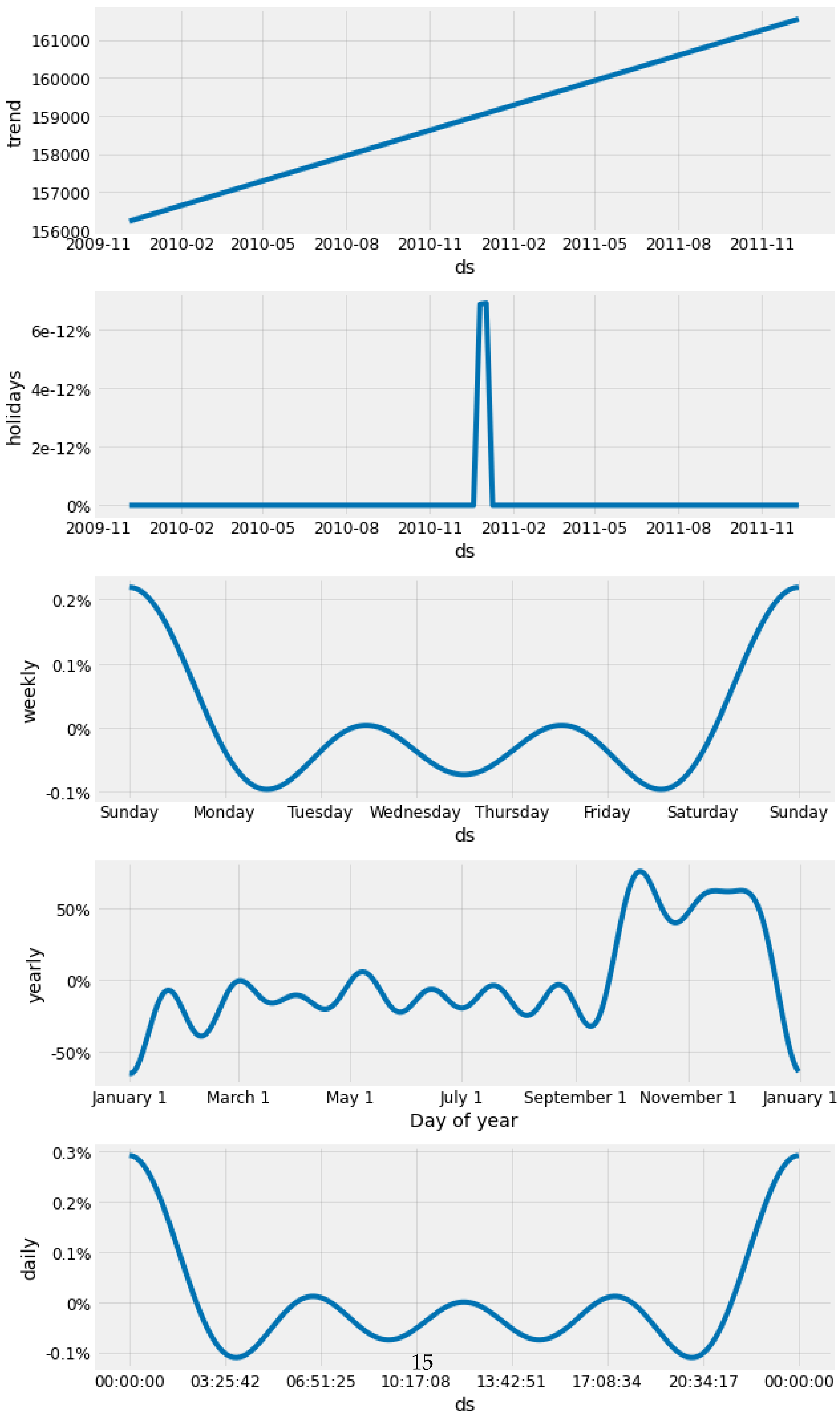
```
r2 train: 0.8167461013856224
r2 test: 0.6539555752829709
------------------------------
mae train: 18666.732750819254
mae test: 28783.227668506217
```

[31]:

```
fig = model_3.plot(forecast_3)
ax = fig.add_subplot(111)
ax.axvline(x=forecast_3['ds'].max() - pd.Timedelta('371 days'), c='red', lw=2,␣
 ↪alpha=0.5, ls='--')
fig.show()
```



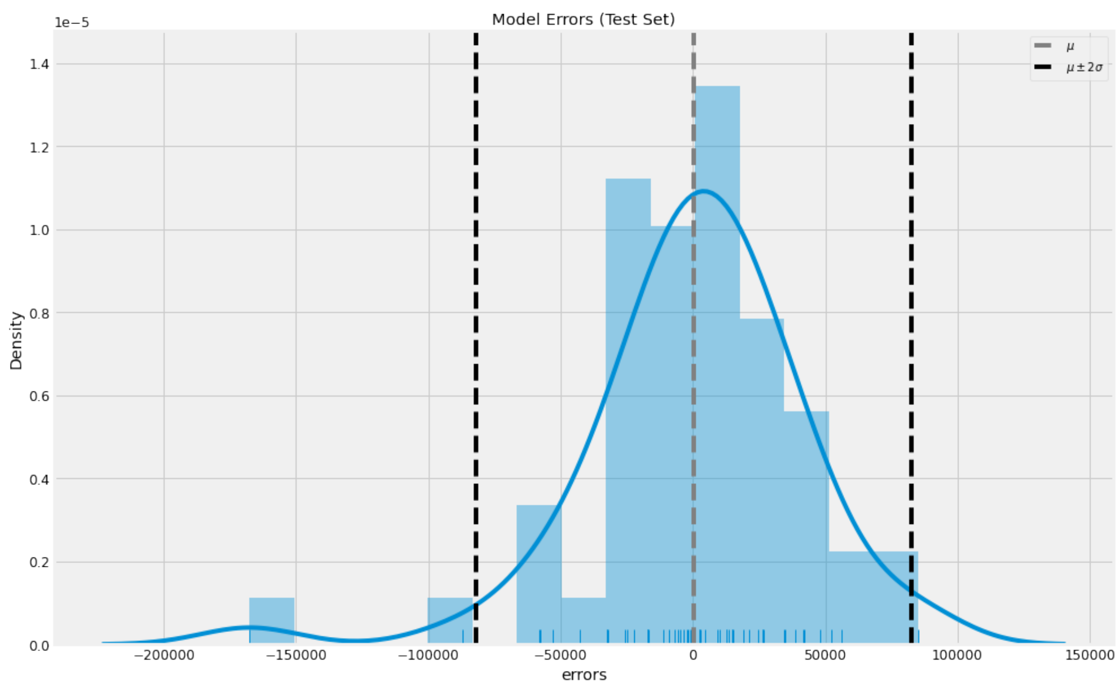
[32]:
```
fig = model_3.plot_components(forecast_3);
```


161000
160000
159000
158000
157000
156000
trend
2009-11
2010-02
2010-05
2010-08
2010-11
2011-02
2011-05
2011-08
2011-11
ds
6e-12%
4e-12%
2e-12%
0%
holidays
2009-11
2010-02
2010-05
2010-08
2010-11
2011-02
2011-05
2011-08
2011-11
ds
0.2%
0.1%
0%
-0.1%
weekly
Sunday
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday
ds
50%
0%
-50%
yearly
January 1
March 1
May 1
July 1
September 1
November 1
January 1
Day of year
0.3%
0.2%
0.1%
0%
-0.1%
daily
00:00:00
03:25:42
06:51:25
10:17:08
13:42:51
17:08:34
20:34:17
00:00:00
ds

After re-applying the Prophet model taking into account the holidays effect, no differences are noticed compared to the second implementation. By plotting the components of the model, an increasing trend is noticed as well as seasonality patterns.

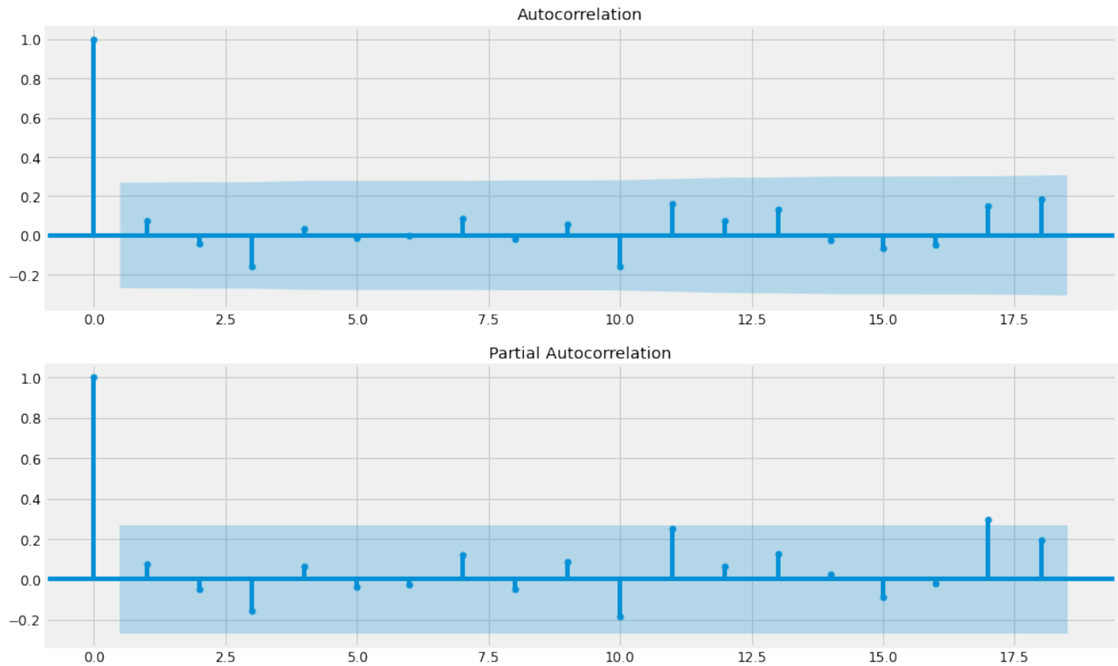
Let us also have a look at the error's values and distribution.

[33]:
```
forecast_test_2.loc[:, 'errors'] = forecast_test_2.loc[:, 'yhat'] -␣
↪test_weekly_data.loc[:, 'y']

rcParams['figure.figsize'] = 15, 10

errors_mean = forecast_test_2['errors'].mean()
errors_std = forecast_test_2['errors'].std()

fig, ax = plt.subplots()

sns.distplot(a=forecast_test_2['errors'], ax=ax, bins=15, rug=True)
ax.axvline(x=errors_mean, color='grey', linestyle='--', label=r'$\mu$')
ax.axvline(x=errors_mean + 2*errors_std, color='black', linestyle='--',␣
↪label=r'$\mu \pm 2\sigma$')
ax.axvline(x=errors_mean - 2*errors_std, color='black', linestyle='--')
ax.legend()
ax.set(title='Model Errors (Test Set)');
```



[34]:

```python
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

fig, ax = plt.subplots(2, 1)
plot_acf(x=forecast_test_2['errors'], ax=ax[0])
plot_pacf(x=forecast_test_2['errors'], ax=ax[1]);
```



By investigating the autocorrelation, it is clear that there is no (partial) autocorrelation.

[ ]:

```python

```

[35]:

```python
# Extend dates and features.
future_2 = model_2.make_future_dataframe(test_weekly_data.shape[0] + 4, freq='W')
```

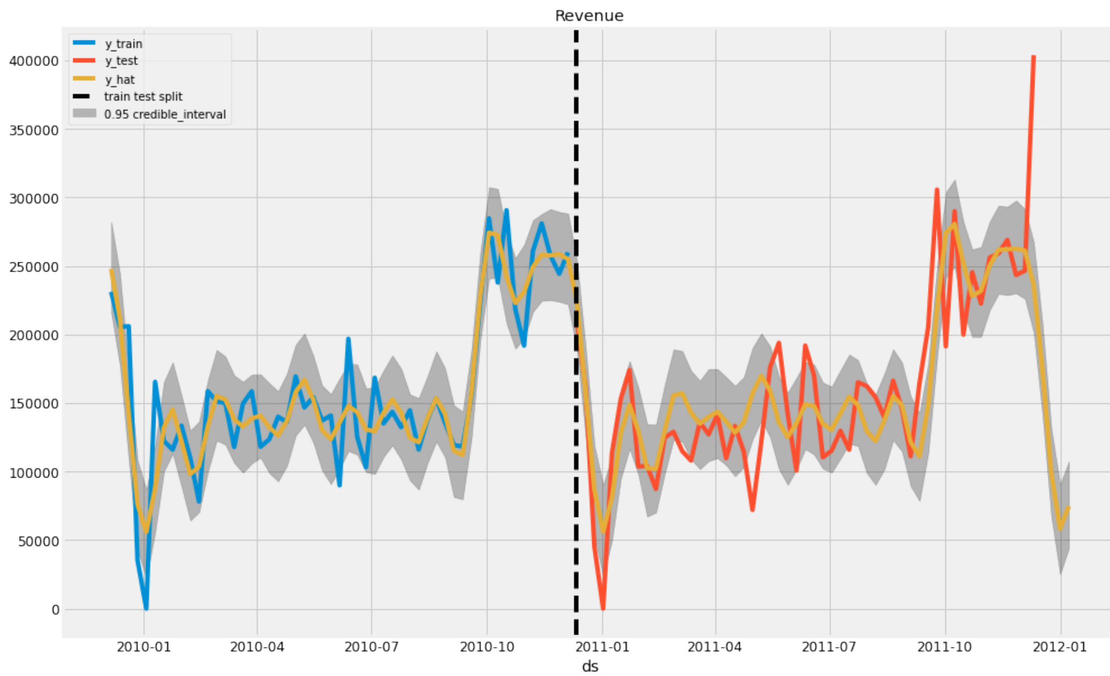
[36]:

```python
# Generate predictions.
forecast_2 = model_2.predict(future_2)
```

[37]:

```python
fig, ax = plt.subplots()

from pylab import rcParams
rcParams['figure.figsize'] = 15, 10

ax.fill_between(
    x=forecast_2['ds'],
    y1=forecast_2['yhat_lower'],
    y2=forecast_2['yhat_upper'],
```

```
    color='black',
    alpha = 0.25,
    label=r'0.95 credible_interval'
)

sns.lineplot(x='ds', y='y', label='y_train', data=train_weekly_data, ax=ax)
sns.lineplot(x='ds', y='y', label='y_test', data=test_weekly_data, ax=ax)
sns.lineplot(x='ds', y='yhat', label='y_hat', data=forecast_2, ax=ax)
ax.axvline(threshold_date, color='black', linestyle='--', label='train test␣
 ↪split')
ax.legend(loc='upper left')
ax.set(title='Revenue', ylabel='');
```



[38]:
```
forecast_2.iloc[106:,:]
```

[38]:
```
            ds          trend     yhat_lower     yhat_upper    trend_lower  \
106 2011-12-18  161597.753361  141254.996903  207443.715073  161597.737543
107 2011-12-25  161648.448713   67300.153090  131532.426020  161648.432527
108 2012-01-01  161699.144064   25146.613068   90762.776301  161699.127381
109 2012-01-08  161749.839415   44090.052676  107459.073113  161749.822205

       trend_upper     daily  daily_lower  daily_upper  multiplicative_terms  \
106  161597.768126  0.002919     0.002919     0.002919              0.078911
107  161648.464134  0.002919     0.002919     0.002919             -0.385659
108  161699.159905  0.002919     0.002919     0.002919             -0.640306
```

```
109  161749.855848  0.002919     0.002919     0.002919              -0.538136

      ...     weekly  weekly_lower  weekly_upper    yearly  yearly_lower  \
106  ...  0.002189      0.002189      0.002189  0.073803      0.073803
107  ...  0.002189      0.002189      0.002189 -0.390767     -0.390767
108  ...  0.002189      0.002189      0.002189 -0.645414     -0.645414
109  ...  0.002189      0.002189      0.002189 -0.543244     -0.543244

     yearly_upper  additive_terms  additive_terms_lower  additive_terms_upper  \
106      0.073803             0.0                   0.0                   0.0
107     -0.390767             0.0                   0.0                   0.0
108     -0.645414             0.0                   0.0                   0.0
109     -0.543244             0.0                   0.0                   0.0

              yhat
106  174349.536859
107   99307.314620
108   58162.249164
109   74706.425506

[4 rows x 22 columns]
```

[39]:
```
forecast_2[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].iloc[106:,:]
```

[39]:
```
             ds           yhat     yhat_lower     yhat_upper
106  2011-12-18  174349.536859  141254.996903  207443.715073
107  2011-12-25   99307.314620   67300.153090  131532.426020
108  2012-01-01   58162.249164   25146.613068   90762.776301
109  2012-01-08   74706.425506   44090.052676  107459.073113
```

Then, re-applying the second model by extending the desired predictive time period to 4 more weeks, the main target of the project is achieved. In the above plot, the future pattern of the weekly revenue is noticed, where a steep decline is predicted from the model, which appears to be in order with the past years where a 0 value is observed at the 1st day of the year.

[ ]:
```

```

Finally, cross-validation is performed to measure forecast error and assess prediction on a horizon of 28 days, starting with 220 days of training datain the first cutoof and then make predictions every 7 days.

[40]:
```
df_cv = cross_validation(model_2, initial='220 days', period='7 days', horizon =␣
↪'28 days')
```

```
INFO:fbprophet:Making 17 forecasts with cutoffs between 2010-07-18 00:00:00 and
2010-11-07 00:00:00
WARNING:fbprophet:Seasonality has period of 365.25 days which is larger than
initial window. Consider increasing initial.
```

```
  0%|          | 0/17 [00:00<?, ?it/s]
```

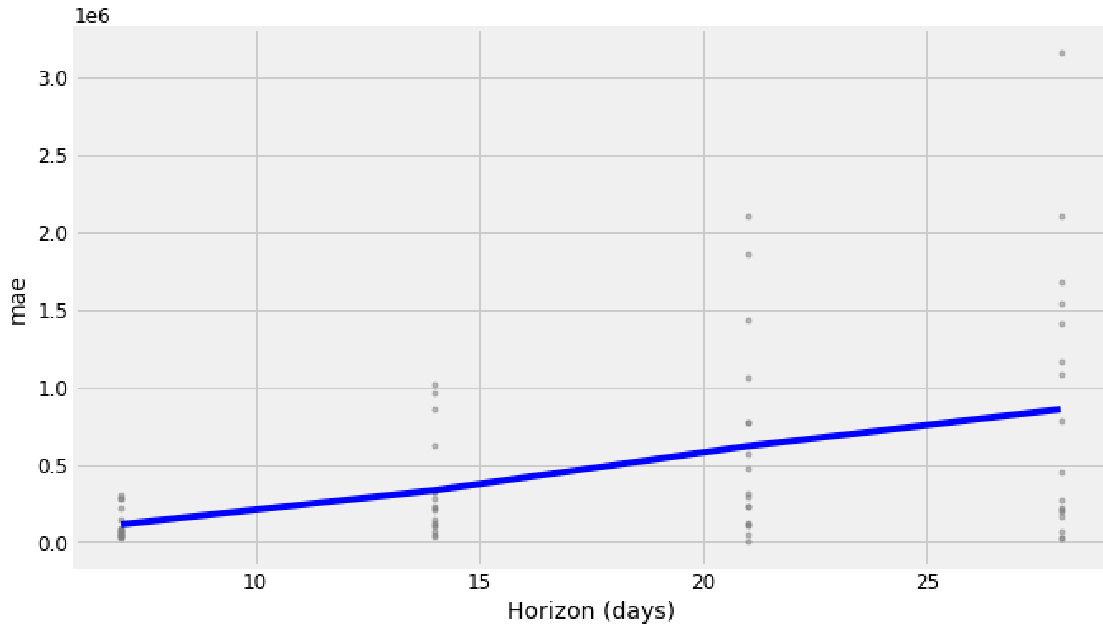
[41]:
```
df_cv
```

[41]:
```
           ds          yhat    yhat_lower    yhat_upper          y     cutoff
0  2010-07-25  5.192892e+04  1.879248e+04  8.008791e+04  132441.41 2010-07-18
1  2010-08-01 -2.426988e+03 -3.538803e+04  3.000902e+04  144686.03 2010-07-18
2  2010-08-08  6.845892e+04  3.877906e+04  1.022770e+05  116033.59 2010-07-18
3  2010-08-15  1.611541e+05  1.298257e+05  1.911832e+05  137896.77 2010-07-18
4  2010-08-01  3.612569e+05  3.296247e+05  3.911384e+05  144686.03 2010-07-25
..        ...           ...           ...           ...        ...        ...
63 2010-11-28  4.656921e+04  1.521633e+04  7.914370e+04  244281.39 2010-10-31
64 2010-11-14  5.681425e+05  5.347622e+05  5.990410e+05  280985.77 2010-11-07
65 2010-11-21  1.114965e+06  1.084747e+06  1.148262e+06  257536.70 2010-11-07
66 2010-11-28  1.682976e+06  1.649881e+06  1.714681e+06  244281.39 2010-11-07
67 2010-12-05  1.935288e+06  1.903756e+06  1.967053e+06  259911.72 2010-11-07

[68 rows x 6 columns]
```

[42]:
```
df_p = performance_metrics(df_cv)
df_p
```

[42]:
```
  horizon           mse          rmse            mae      mape     mdape  \
0  7 days  2.144704e+10  1.464481e+05  113036.263971  0.621709  0.499531
1 14 days  2.104791e+11  4.587800e+05  333395.765450  1.892807  1.016774
2 21 days  7.740137e+11  8.797805e+05  618476.926673  3.541355  2.617182
3 28 days  1.484299e+12  1.218318e+06  856996.268880  4.708285  1.588319

   coverage
0  0.000000
1  0.000000
2  0.058824
3  0.117647
```

[43]:
```
fig = plot_cross_validation_metric(df_cv, metric='mae')
```


1e6
3.0
2.5
2.0
1.5
1.0
0.5
0.0
mae
10
15
20
25
Horizon (days)