In [4]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from prophet import Prophet
```

In [51]:
```
from sklearn.metrics import mean_squared_error, mean_absolute_error
import warnings
warnings.filterwarnings("ignore")
plt.style.use("ggplot")
plt.style.use("fivethirtyeight")

def mean_absolute_percentage_error(y_true, y_pred):
    """ CALCULATE MAPE given y_true and y_pred """
    y_true,y_pred=np.array(y_true),np.array(y_pred)
    return np.mean(np.abs((y_true,y_pred)/y_true))*100
```

In [52]:
```
sales = pd.read_csv("Sample - Superstore.csv", encoding="latin1",index_col=[0]
sales.head()
```

Out[52]:

Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City
1	CA-2016-152156	11/8/2016	11/11/2016	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson
2	CA-2016-152156	11/8/2016	11/11/2016	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson
3	CA-2016-138688	6/12/2016	6/16/2016	Second Class	DV-13045	Darrin Van Huff	Corporate	United States	Los Angeles
4	US-2015-108966	10/11/2015	10/18/2015	Standard Class	SO-20335	Sean O'Donnell	Consumer	United States	Fort Lauderdale
5	US-2015-108966	10/11/2015	10/18/2015	Standard Class	SO-20335	Sean O'Donnell	Consumer	United States	Fort Lauderdale

In [53]:
```
sales['Order Date'] = pd.to_datetime(sales['Order Date'])
sales['Ship Date'] = pd.to_datetime(sales['Ship Date'])
```

In [54]:
```
sales.dtypes
```

Out[54]:
```
Order ID                  object
Order Date        datetime64[ns]
Ship Date         datetime64[ns]
```

```
Ship Mode              object
Customer ID            object
Customer Name          object
Segment                object
Country                object
City                   object
State                  object
Postal Code             int64
Region                 object
Product ID             object
Category               object
Sub-Category           object
Product Name           object
Sales                 float64
Quantity                int64
Discount              float64
Profit                float64
dtype: object
```

In [102…

```
 df_sales=sales[['Order Date','Sales']]
df_sales.head()
```

Out[102]:

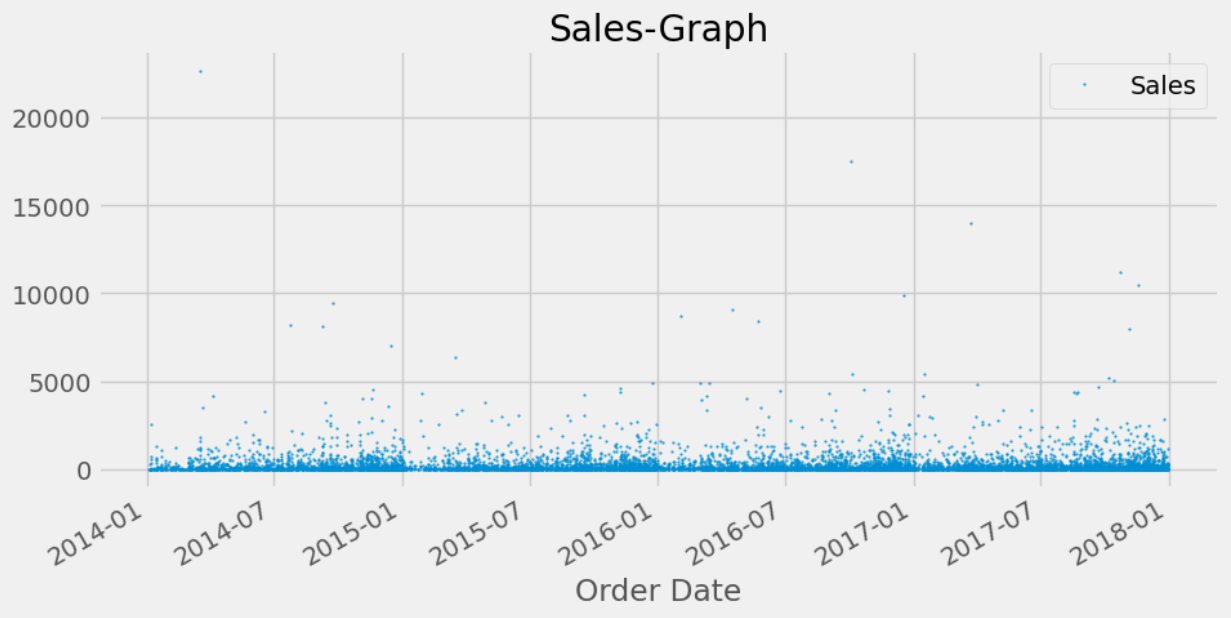
Row ID	Order Date	Sales
1	2016-11-08	261.9600
2	2016-11-08	731.9400
3	2016-06-12	14.6200
4	2015-10-11	957.5775
5	2015-10-11	22.3680

In [103…

```
df_sales.set_index('Order Date',inplace=True)
color_pal=sns.color_palette()
df_sales.plot(style='.',
       figsize=(10,5),
       ms=1,
       color=color_pal[0],
       title='Sales-Graph')
```

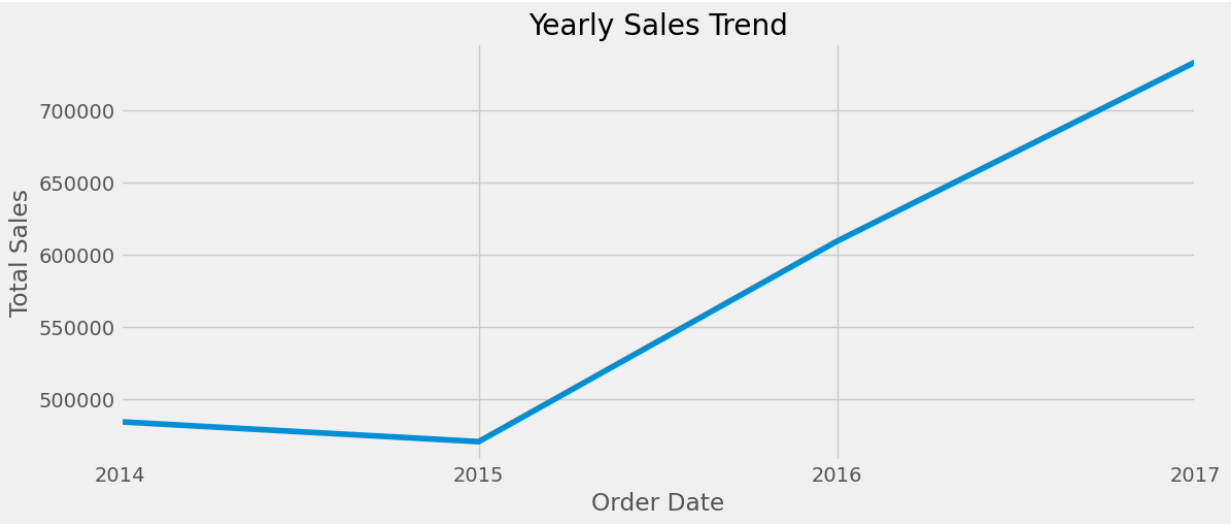
Out[103]:

```
<Axes: title={'center': 'Sales-Graph'}, xlabel='Order Date'>
```



Time Series FEATURES

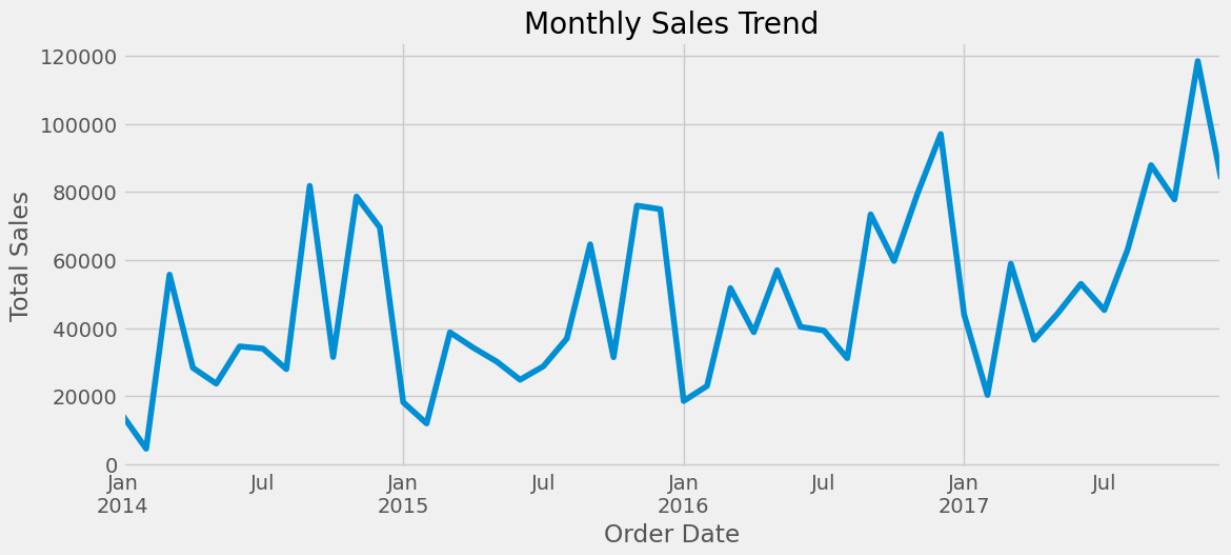
In [104…
```
yearly_sales = df_sales['Sales'].resample('Y').sum()
yearly_sales.plot(figsize=(12,5), title='Yearly Sales Trend')
plt.ylabel('Total Sales')
plt.show()
```



In [112…
```
monthly_sales = df_sales['Sales'].resample('M').sum()
monthly_sales.head()
```

Out[112]:
```
Order Date
2014-01-31    14236.895
2014-02-28     4519.892
2014-03-31    55691.009
2014-04-30    28295.345
2014-05-31    23648.287
Freq: M, Name: Sales, dtype: float64
```

In [113…
```
monthly_sales.plot(figsize=(12,5), title='Monthly Sales Trend')
plt.ylabel('Total Sales')
plt.show()
```



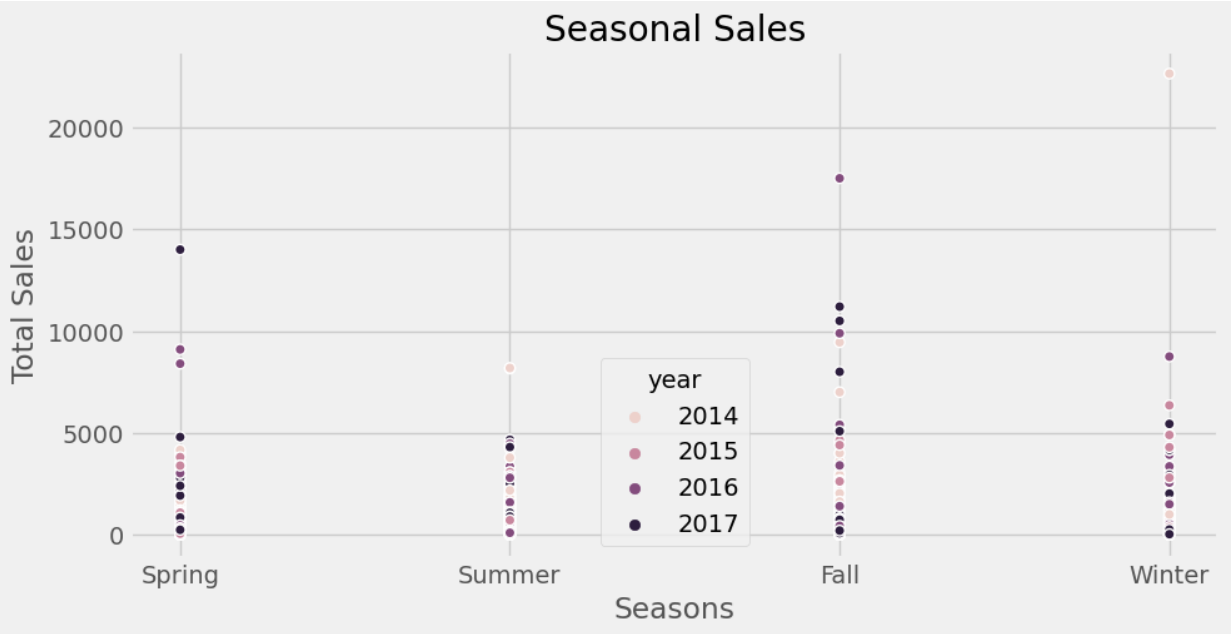
In [107…

```
from pandas.api.types import CategoricalDtype

cat_type = CategoricalDtype(categories=['Monday','Tuesday',
                                        'Wednesday',
                                        'Thursday','Friday',
                                        'Saturday','Sunday'],
                            ordered=True)

def create_features(df, label=None):
    """
    Creates time series features from datetime index.
    """
    df = df.copy()
    df['date'] = df.index
    df['dayofweek'] = df['date'].dt.dayofweek
    df['weekday'] = df['date'].dt.day_name()
    df['weekday'] = df['weekday'].astype(cat_type)
    df['quarter'] = df['date'].dt.quarter
    df['month'] = df['date'].dt.month
    df['year'] = df['date'].dt.year
    df['dayofyear'] = df['date'].dt.dayofyear
    df['dayofmonth'] = df['date'].dt.day
    df['weekofyear'] = df['date'].dt.weekofyear
    df['date_offset'] = (df.date.dt.month*100 + df.date.dt.day - 320)%1300

    df['season'] = pd.cut(df['date_offset'], [0, 300, 602, 900, 1300],
                          labels=['Spring', 'Summer', 'Fall', 'Winter']
                    )
    X = df[['dayofweek','quarter','month','year',
           'dayofyear','dayofmonth','weekofyear','weekday',
           'season']]
    if label:
        y = df[label]
        return X, y
    return X

X, y = create_features(df_sales, label='Sales')
features_and_target = pd.concat([X, y], axis=1)
```

In [109…

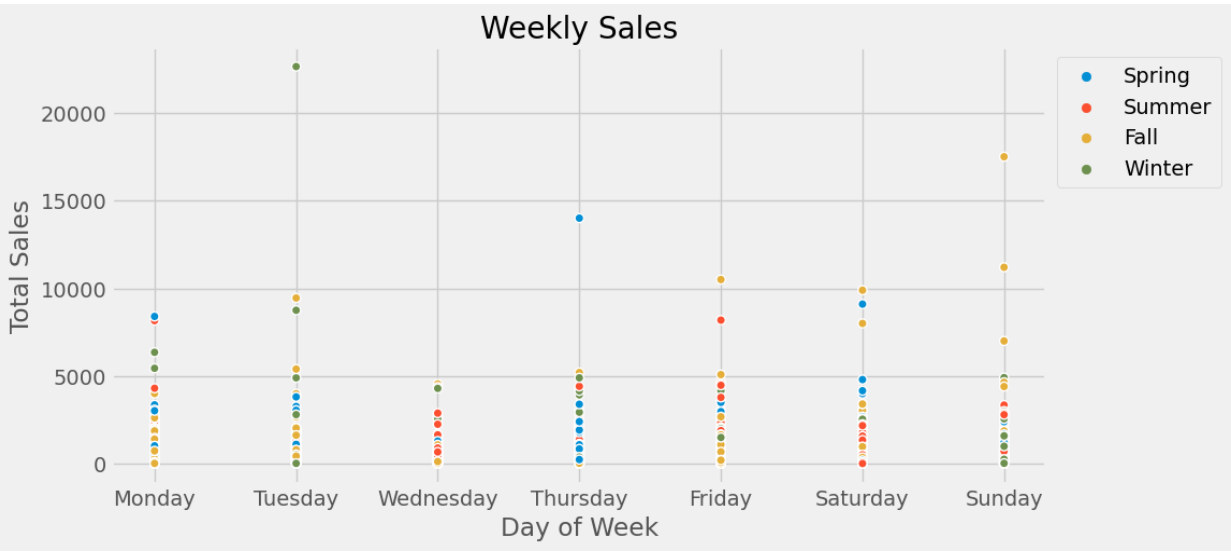
```
fig, ax = plt.subplots(figsize=(10, 5))
sns.scatterplot(data=features_and_target.dropna(),
            x='season',
            y='Sales',
```

```
            hue='year',
            ax=ax,
            linewidth=1)
ax.set_title('Seasonal Sales')
ax.set_xlabel('Seasons')
ax.set_ylabel('Total Sales')
plt.show()
```



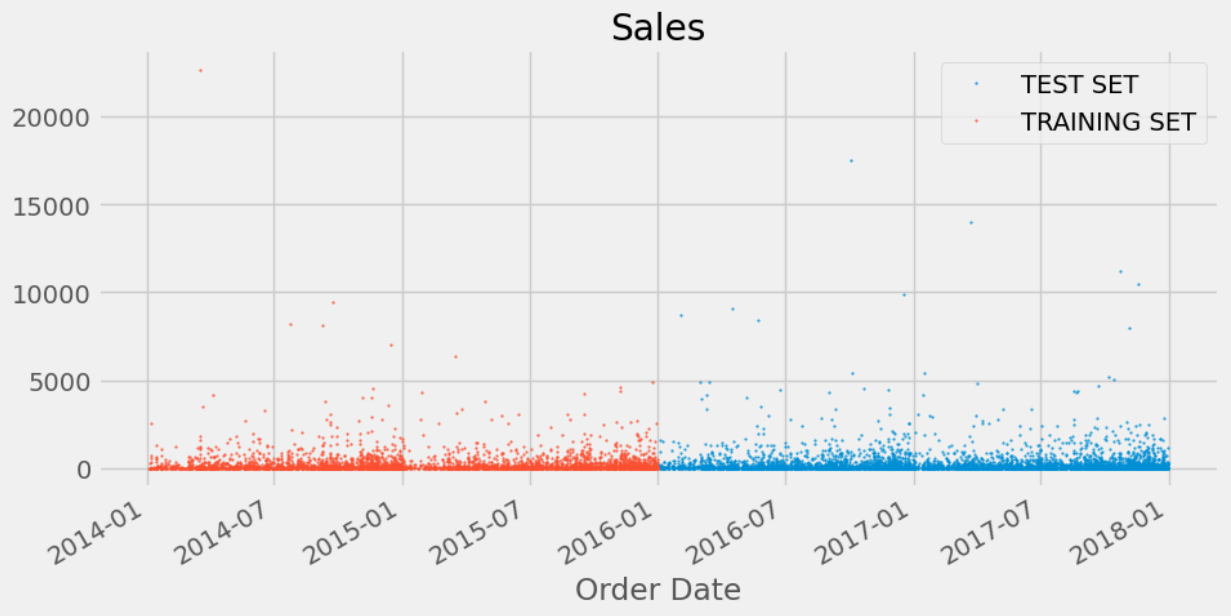
In [108…

```
fig, ax = plt.subplots(figsize=(10, 5))
sns.scatterplot(data=features_and_target.dropna(),
            x='weekday',
            y='Sales',
            hue='season',
            ax=ax,
            linewidth=1)
ax.set_title('Weekly Sales')
ax.set_xlabel('Day of Week')
ax.set_ylabel('Total Sales')
ax.legend(bbox_to_anchor=(1, 1))
plt.show()
```



TRAIN / TEST SPLIT

In [119…]
```
split_date = '1-Jan-2016'
sales_train = df_sales.loc[df_sales.index <= split_date].copy()
sales_test = df_sales.loc[df_sales.index > split_date].copy()

# Plot train and test so you can see where we have split
sales_test \
    .rename(columns={'Sales': 'TEST SET'}) \
    .join(sales_train.rename(columns={'Sales': 'TRAINING SET'}),
          how='outer') \
    .plot(figsize=(10, 5), title='Sales', style='.', ms=1)
plt.show()
```



Prophet Model

In [120…]
```
sales_train_prophet = sales_train.reset_index() \
    .rename(columns={'Order Date':'ds',
                     'Sales':'y'})
```

In [121…]
```
%%time
model = Prophet()
model.fit(sales_train_prophet)
```

```
14:09:20 - cmdstanpy - INFO - Chain [1] start processing
14:09:21 - cmdstanpy - INFO - Chain [1] done processing
CPU times: total: 656 ms
Wall time: 3.54 s
```

Out[121]:
```
<prophet.forecaster.Prophet at 0x256362cea90>
```

In [122…]
```
sales_test_prophet = sales_test.reset_index() \
    .rename(columns={'Order Date':'ds',
                     'Sales':'y'})

sales_test_fcst = model.predict(sales_test_prophet)
```

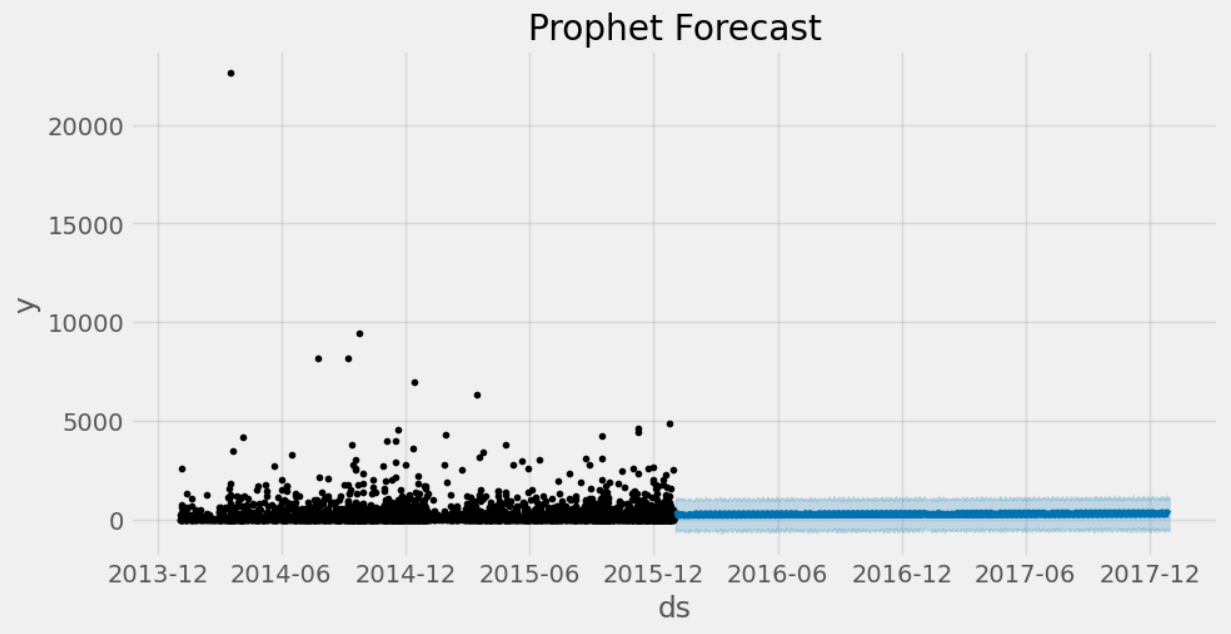
In [124…]
```
sales_test_fcst.head()
```

Out[124]:

	ds	trend	yhat_lower	yhat_upper	trend_lower	trend_upper	additive_terms	additive_
0	2016-01-02	235.436247	-560.119264	1012.173749	235.436247	235.436247	-32.391618	
1	2016-01-02	235.436247	-580.770467	1008.206099	235.436247	235.436247	-32.391618	
2	2016-01-03	235.521072	-559.389765	1029.762484	235.521072	235.521072	-20.934712	
3	2016-01-03	235.521072	-580.362755	1016.507645	235.521072	235.521072	-20.934712	
4	2016-01-03	235.521072	-653.657606	1046.137797	235.521072	235.521072	-20.934712	

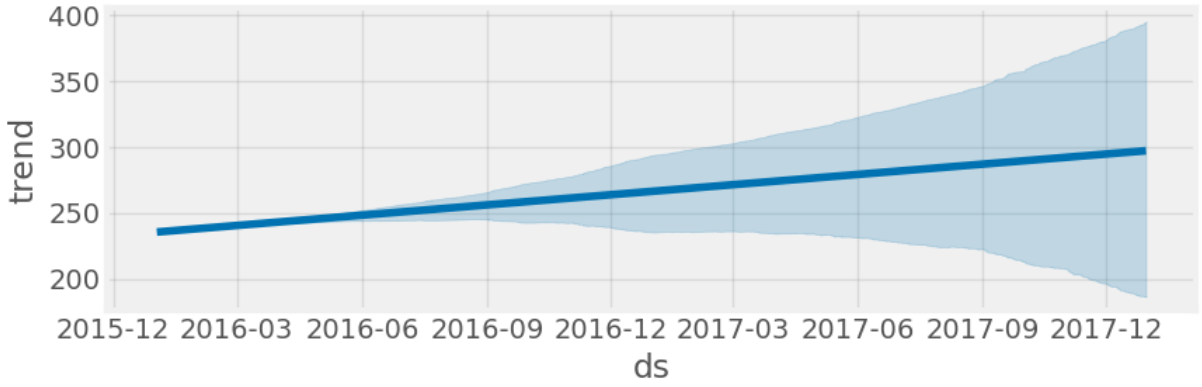
In [125…

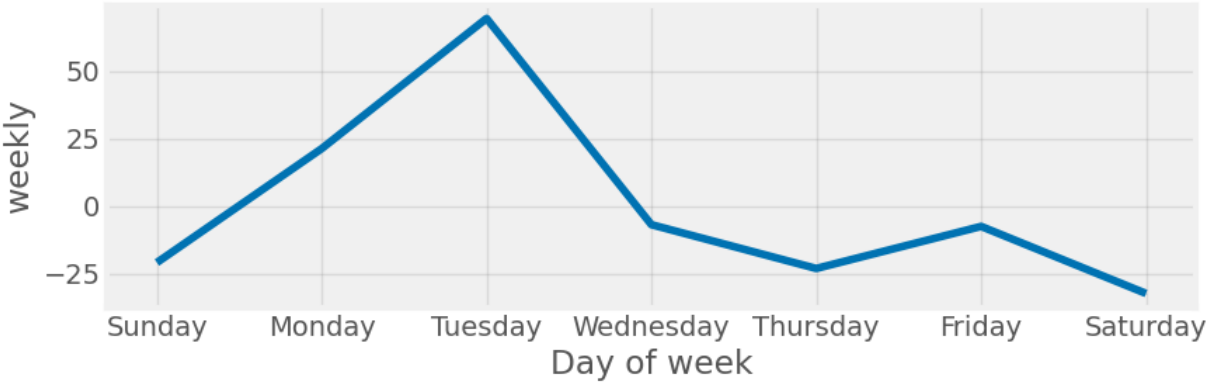
```
fig, ax = plt.subplots(figsize=(10, 5))
fig = model.plot(sales_test_fcst, ax=ax)
ax.set_title('Prophet Forecast')
plt.show()
```



In [126…

```
fig = model.plot_components(sales_test_fcst)
plt.show()
```

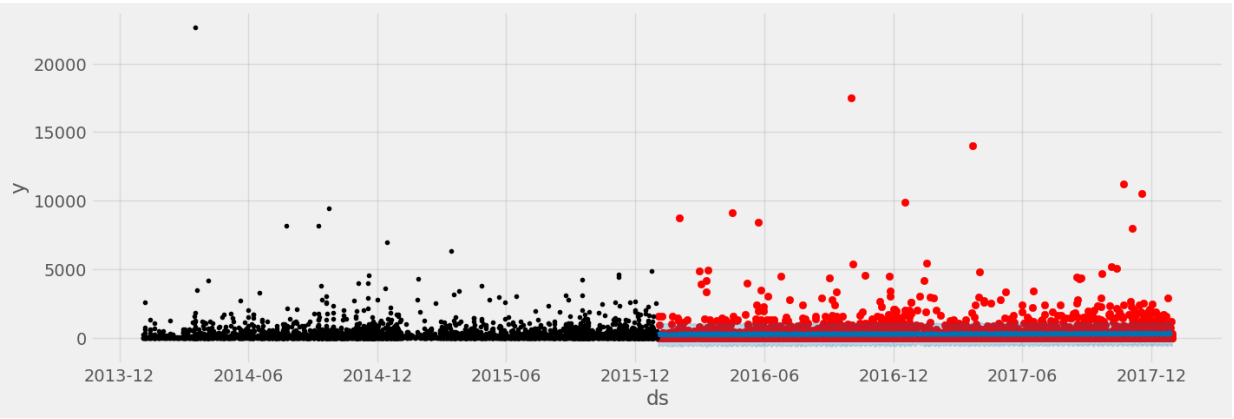




Compare Forecast to Actuals

In [127…

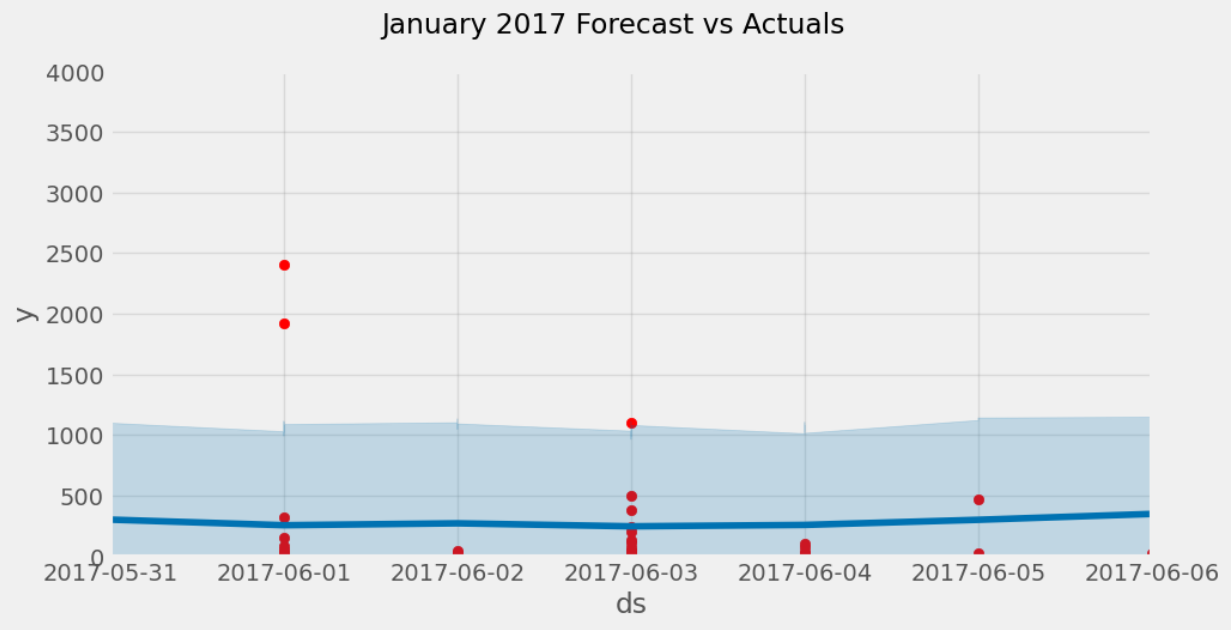
```
f, ax = plt.subplots(figsize=(15, 5))
ax.scatter(sales_test.index, sales_test['Sales'], color='r')
fig = model.plot(sales_test_fcst, ax=ax)
```



In [ ]:

In [147…

```
# Convert the index to datetime objects
fig, ax = plt.subplots(figsize=(10, 5))
ax.scatter(sales_test.index, sales_test['Sales'], color='r')
fig = model.plot(sales_test_fcst, ax=ax)
ax.set_xbound(lower=pd.to_datetime('2017-05-31'),
              upper=pd.to_datetime('2017-06-06'))

ax.set_ylim(0, 4000)
plot = plt.suptitle('January 2017 Forecast vs Actuals')
```



Evaluate with error metrics

In [148…
```
np.sqrt(mean_squared_error(y_true=sales_test['Sales'],
                           y_pred=sales_test_fcst['yhat']))
```

Out[148]:
```
624.7459380197107
```

In [150…
```
mean_absolute_error(y_true=sales_test['Sales'],
                    y_pred=sales_test_fcst['yhat'])
```

Out[150]:
```
289.1339525104914
```

In [151…
```
mean_absolute_percentage_error(y_true=sales_test['Sales'],
                    y_pred=sales_test_fcst['yhat'])
```

Out[151]:
```
718.2798885848571
```

Adding Holidays

In [152…
```
from pandas.tseries.holiday import USFederalHolidayCalendar as calendar

cal = calendar()


holidays = cal.holidays(start=df_sales.index.min(),
                        end=df_sales.index.max(),
                        return_name=True)
holiday_df = pd.DataFrame(data=holidays,
                          columns=['holiday'])
holiday_df = holiday_df.reset_index().rename(columns={'index':'ds'})
```

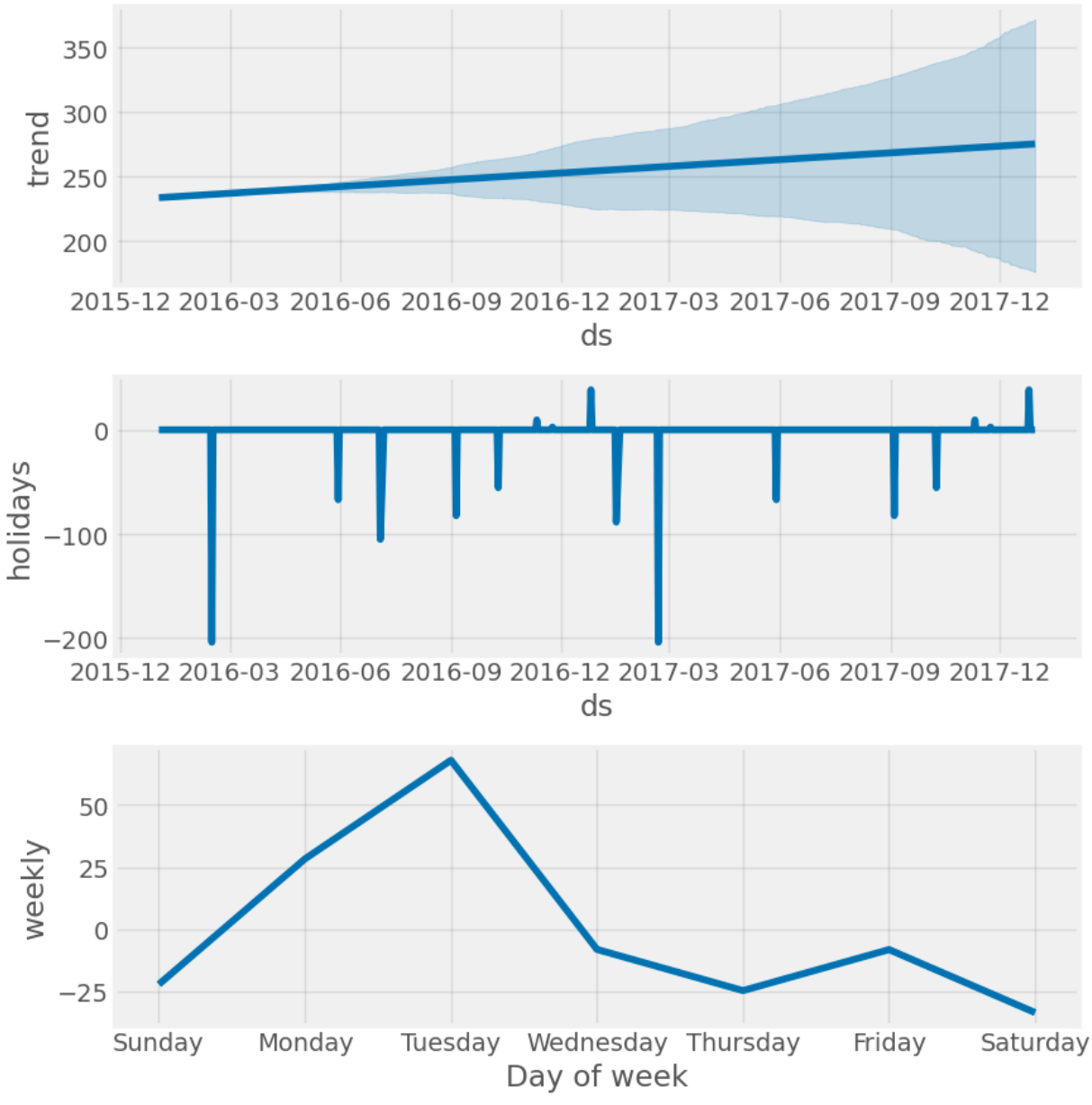
In [154…
```
%%time
model_with_holidays = Prophet(holidays=holiday_df)
model_with_holidays.fit(sales_train_prophet)
```

```
14:37:34 - cmdstanpy - INFO - Chain [1] start processing
14:37:34 - cmdstanpy - INFO - Chain [1] done processing
CPU times: total: 344 ms
Wall time: 921 ms
```

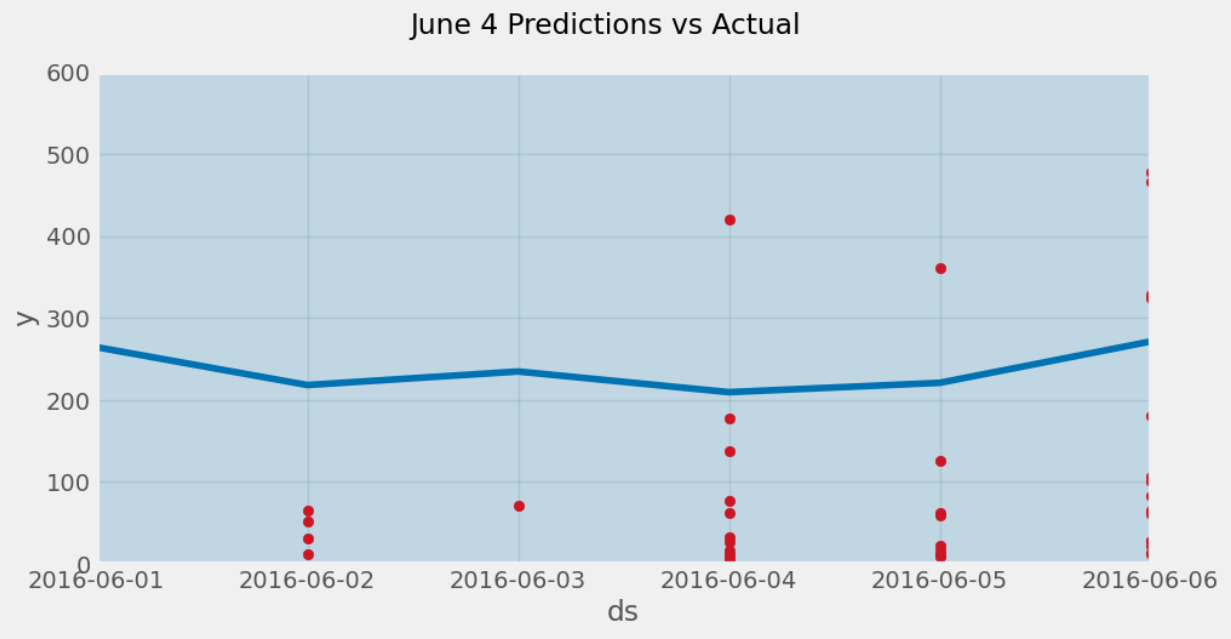
Out[154]:
```
<prophet.forecaster.Prophet at 0x25632168a90>
```

In [155…
```
sales_test_fcst_with_hols = \
    model_with_holidays.predict(df=sales_test_prophet)
```

In [156…
```
fig = model_with_holidays.plot_components(
    sales_test_fcst_with_hols)
plt.show()
```



In [166…
```
fig, ax = plt.subplots(figsize=(10, 5))
ax.scatter(sales_test.index, sales_test['Sales'], color='r')
fig = model.plot(sales_test_fcst_with_hols, ax=ax)
ax.set_xbound(lower=pd.to_datetime('2016-06-01'),
              upper=pd.to_datetime('2016-06-06'))
ax.set_ylim(0,600 )
plot = plt.suptitle('June 4 Predictions vs Actual')
```



In [167…
```
mean_absolute_percentage_error(y_true=sales_test['Sales'],
                               y_pred=sales_test_fcst_with_hols['yhat'])
```

Out[167]:
```
680.2267281973047
```

Predict to the future

In [168…
```
future = model.make_future_dataframe(periods=365*24, freq='h', include_history
forecast = model_with_holidays.predict(future)
```

In [169…
```
forecast[['ds','yhat']].head()
```

Out[169]:

	ds	yhat
0	2015-12-31 01:00:00	210.507067
1	2015-12-31 02:00:00	212.032597
2	2015-12-31 03:00:00	213.577754
3	2015-12-31 04:00:00	215.122866
4	2015-12-31 05:00:00	216.648851