TimeSeries_Forecasting_Project

April 5, 2019

Table of Contents

1 Introduction: Time Series Forecasting Project

In this notebook, I will demonstrate how to implement a time series forecasting project based on a historical dataset of sales.

1.1 Dataset

The dataset is a historical data of sales stored at datasets folder.

1.2 Python Library

I will use Python library **pandas** & **numpy** to read-in and process the data from a local machine, **matplotlib** & **seaborn** to visualize the data and forecasting results, **statmodels** and **fbprophet** for time series forecasting approaches. For the approaches, I will examine and evaluate **Seasonal**

AutoRegression Integrated Moving Average (SARIMA), **AutoRegression Integrated Moving Average (ARIMA)**, **Exponential Smoothing** and **Facebook Prophet** approaches.

```
In [1]: # Pandas and numpy for data processing
        import pandas as pd
        import numpy as np
        # from pandas.tseries.offsets import MonthEnd
        # Make the random numbers predictable
        np.random.seed(42)
        from sklearn.metrics import mean_squared_error, mean_absolute_error, median_absolute_e
```

```
In [2]: # Time Series Models
        # import FB Prophet
        from fbprophet import Prophet
        from statsmodels.tsa.arima_model import ARIMA
        from statsmodels.tsa.statespace.sarimax import SARIMAX
        from statsmodels.tsa.holtwinters import ExponentialSmoothing
```

```
In [10]: # Matplotlib and seaborn for visualization
         import matplotlib.pyplot as plt
         import matplotlib
         import seaborn as sns

         # Set up matplotlib environment
         %matplotlib inline
         matplotlib.rcParams['font.size'] = 12
         matplotlib.rcParams['figure.figsize'] = (18, 18)

         from IPython.core.pylabtools import figsize
```

2 Exploratory Data Loading

2.1 Read In Data From csv Files

```
In [6]: # Read three data sets
        # need a different encoding rather than the default utf-8
        # due to some specific letters in the data sets
        df = pd.read_csv('../datasets/forecasting_data_RAV4_sales.csv')
        df = df.loc[(df['sales'] >0), ['Month', 'sales']]
        df.rename(columns={'Month':'ds', 'sales':'y'}, inplace=True)
        df.index = pd.to_datetime(df.ds)
        df.sort_index(inplace=True)
        df.head(10)
```

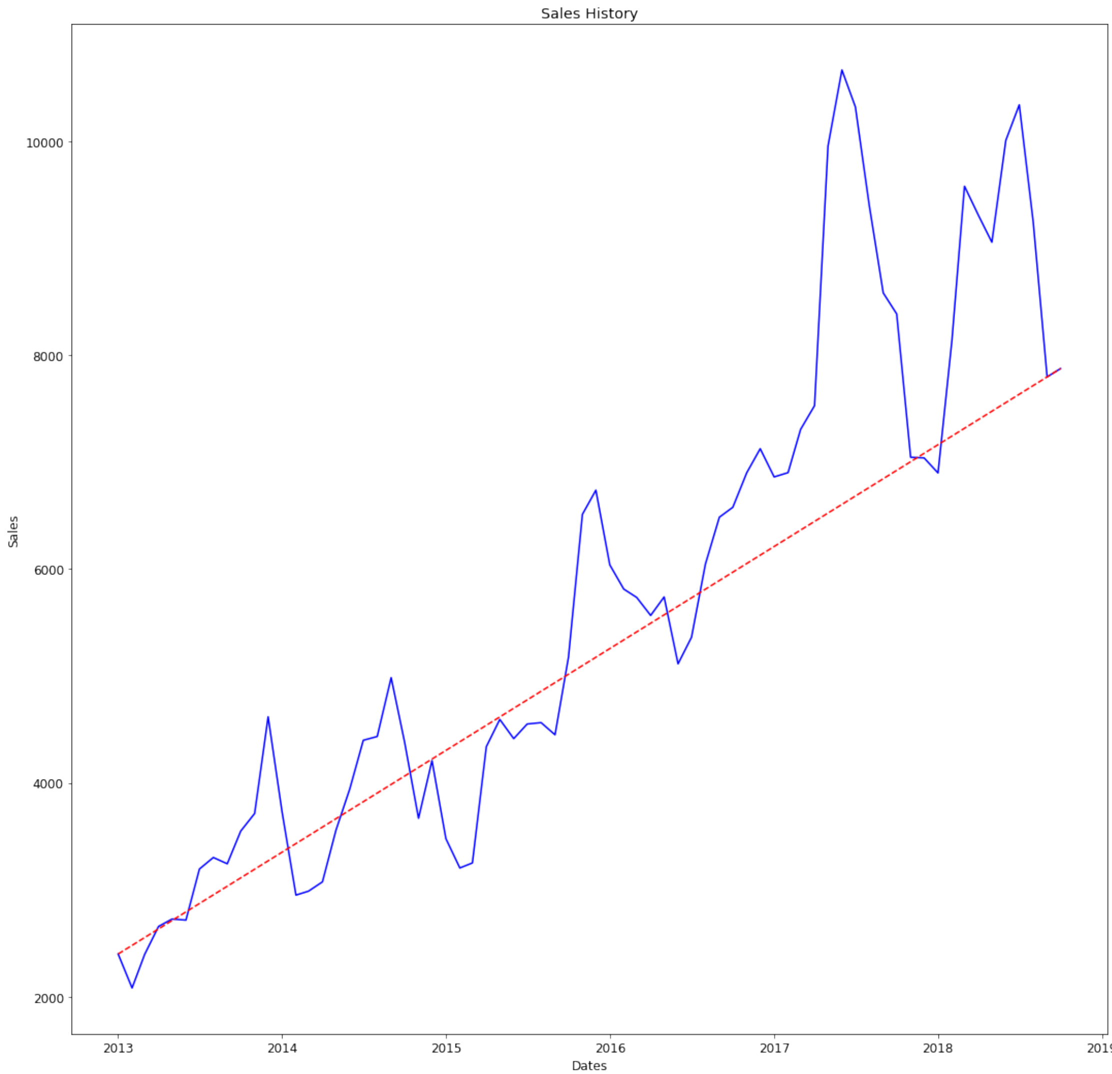
```
Out[6]:                    ds     y
        ds
        2013-01-01   1/1/2013  2399
        2013-02-01   2/1/2013  2081
```

```
2013-03-01   3/1/2013  2395
2013-04-01   4/1/2013  2657
2013-05-01   5/1/2013  2726
2013-06-01   6/1/2013  2716
2013-07-01   7/1/2013  3194
2013-08-01   8/1/2013  3302
2013-09-01   9/1/2013  3243
2013-10-01  10/1/2013  3549
```

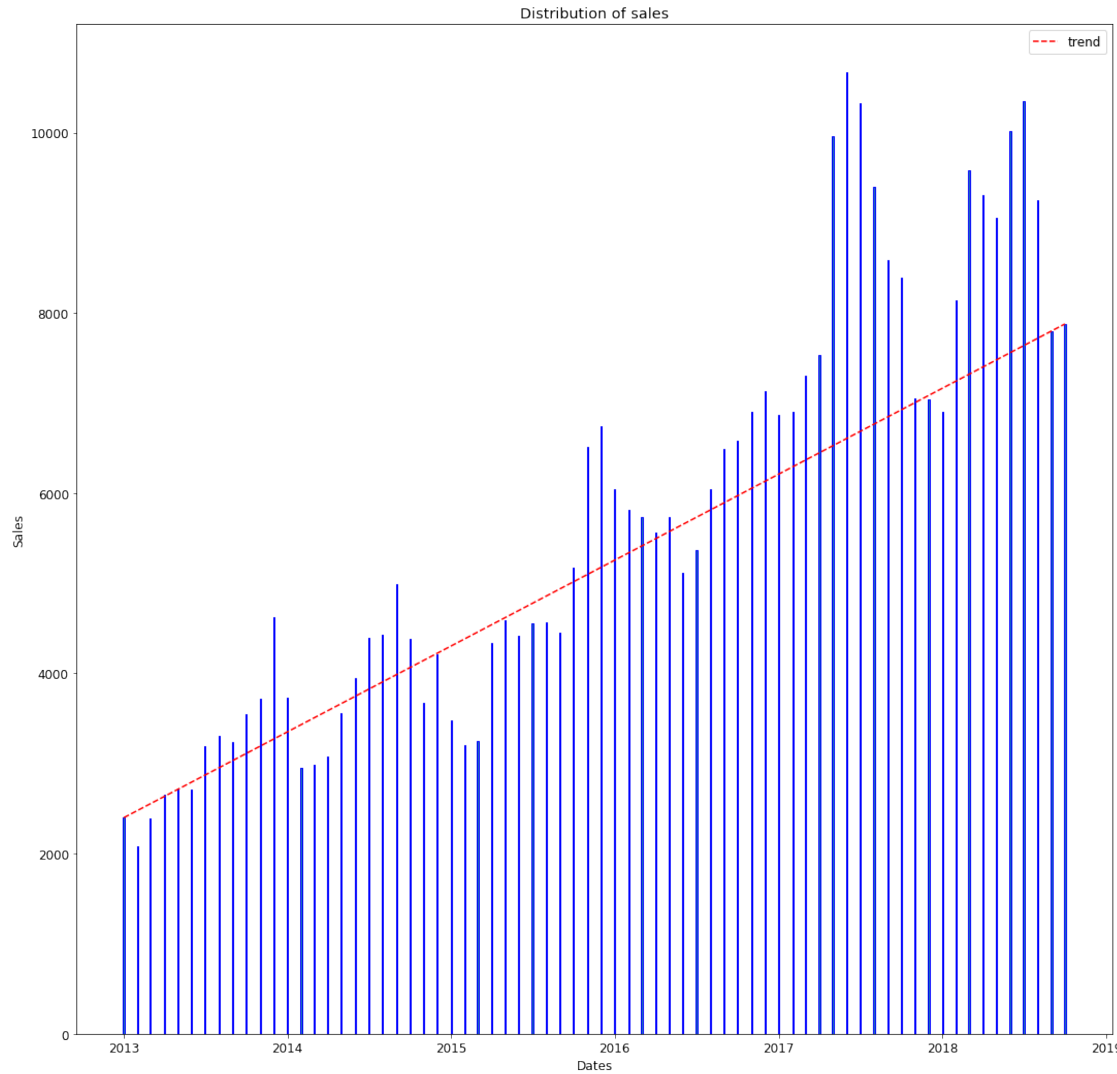
2.2 Distribution and Trending of Sales Data

```
In [11]: # Trending plot of sales
         plt.plot(df.index, df['y'].values, color='blue', linestyle='-')
         plt.plot([df.index[0], df.index[-1]], [df['y'].values[0], df['y'].values[-1]], label=
         plt.xlabel('Dates')
         plt.ylabel('Sales')
         plt.title('Sales History')
```

```
Out[11]: Text(0.5, 1.0, 'Sales History')
```




```
In [12]: # Bar plot of sales
         plt.bar(df.index, df['y'].values, fill = 'blue', edgecolor = 'b', width = 3)
         plt.plot([df.index[0], df.index[-1]], [df['y'].values[0], df['y'].values[-1]], label=
         plt.xlabel('Dates')
         plt.ylabel('Sales')
         plt.title('Distribution of sales')
         plt.legend(loc='upper right')

Out[12]: <matplotlib.legend.Legend at 0x7fd3f034c518>
```



3 Time Series Analytics with Four Approaches

3.1 Setup Metrics

For this analytics project, I will use two standard metrics (wiki definitions) from :

- Mean Absolute Error (MAE): is an interpretable & scale-dependent accuracy measure of difference between two continuous variables and is average vertical distance between each point and the identity line.
- Root Mean Squared Error (RMSE): is a frequently used & scale-dependent measure of the differences between values (sample or population values) predicted by a model or an estimator and the values observed. It represents the square root of the second sample moment of the differences between predicted values and observed values or the quadratic mean of these differences. RMSE is always non-negative, and a value of 0 (almost never achieved in

practice) would indicate a perfect fit to the data. In general, a lower RMSE is better than a higher one.

For more information and discussions around those two metrics, here is a discussion.

3.2 Setup Baseline

For a time series forecasting approach, a simple baseline is to guess the median value on the training set for all testing cases. I will evaluate if forecasting approach can be better than the simple baseline.

```
In [13]: # Baseline is the median
         baseline_pred = df['y'].median()
         baseline_preds = [baseline_pred for _ in range(len(df))]
         real = df['y']
```

```
In [15]: # Function to calculate mae and rmse
         def evaluate_predictions(predictions, real):
             mae = np.mean(abs(predictions - real))
             rmse = np.sqrt(np.mean((predictions - real) ** 2))
             return mae, rmse
```

```
In [16]: mb_MAE, mb_RMSE = evaluate_predictions(baseline_preds, real)
         print('Baseline  MAE: {:.4f}'.format(mb_MAE))
         print('Baseline RMSE: {:.4f}'.format(mb_RMSE))
```

```
Baseline  MAE: 1984.2571
Baseline RMSE: 2379.0198
```

3.3 Setup Training and Testing DataSets

```
In [17]: series = pd.read_csv('../datasets/forecasting_data_RAV4_sales.csv', header=0, parse_da
         # print(series)
         X = series.values
         size = int(len(X) * 0.70)
         train, test = X[0:size], X[size:len(X)]
```

3.4 Seasonal Autoregressive Integrated Moving Average (SARIMA)

SARIMA models are a general time series model, and is used to analyze and forecast data which have an additional seasonal component.

```
In [18]: history = [x for x in train]
         predictions = list()
         for t in range(len(test)):
             model = SARIMAX(history, seasonal_order=(0,0,0,12), enforce_stationarity=False)
             model_fit = model.fit()
             output = model_fit.forecast()
```

```python
            yhat = output[0]
            predictions.append(yhat)
            obs = test[t]
            history.append(obs)
            print('predicted=%f, expected=%f' % (yhat, obs))
        SARIMA_RMSE = np.sqrt(mean_squared_error(test, predictions))
        SARIMA_MAE = mean_absolute_error(test, predictions)
        print('Test MAE: %.3f' % SARIMA_MAE)
        print('Test RMSE: %.3f' % SARIMA_RMSE)
        # plot
        plt.plot(test)
        plt.plot(predictions, color='red')
        plt.show()
```

```
predicted=6965.420380, expected=6903.000000
predicted=7003.222971, expected=7308.000000
predicted=7428.005561, expected=7530.000000
predicted=7658.492329, expected=9959.000000
```

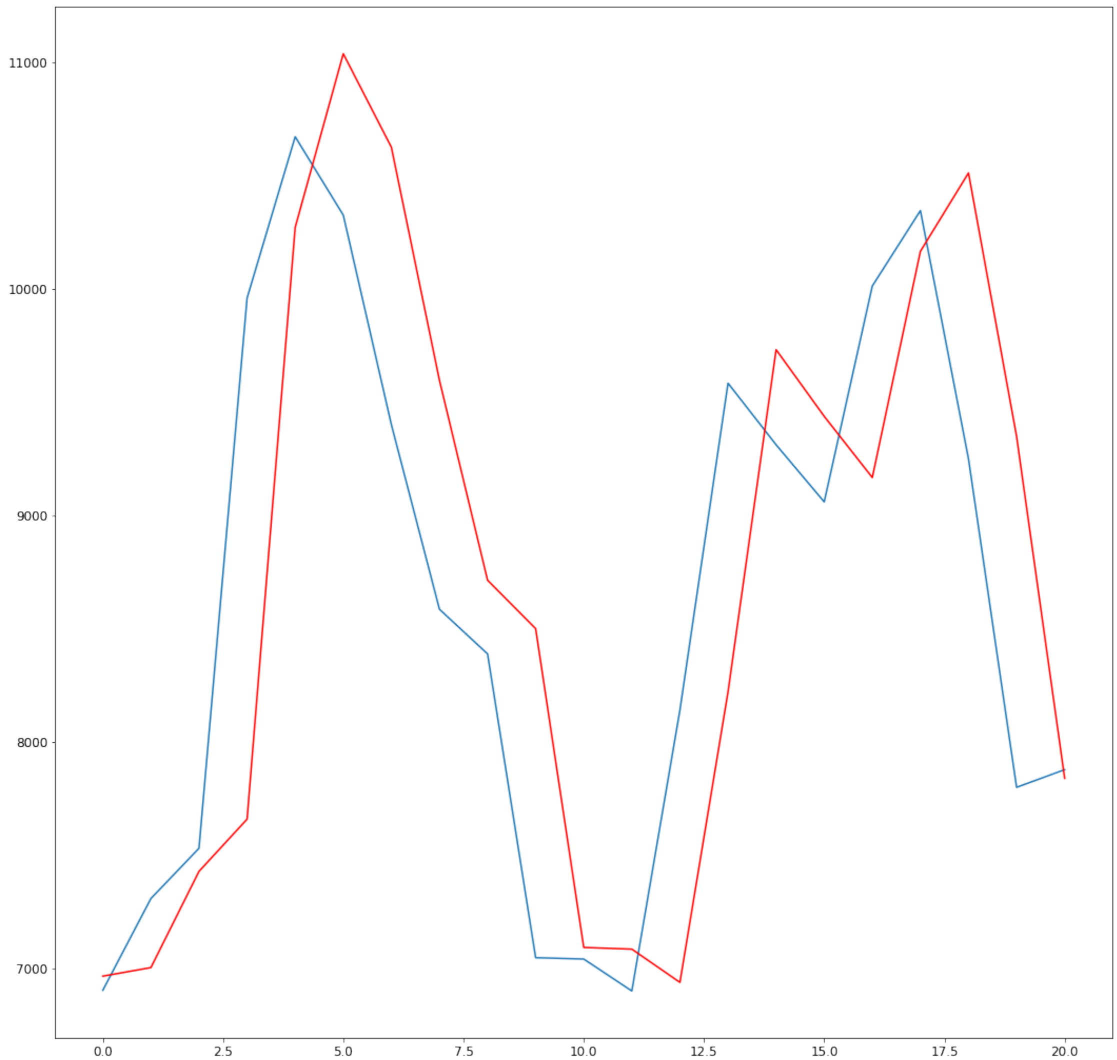
```
/usr/local/lib/python3.6/dist-packages/statsmodels/base/model.py:508: ConvergenceWarning: Maxi
  "Check mle_retvals", ConvergenceWarning)
/usr/local/lib/python3.6/dist-packages/statsmodels/base/model.py:508: ConvergenceWarning: Maxi
  "Check mle_retvals", ConvergenceWarning)
```

```
predicted=10270.810232, expected=10671.000000
predicted=11037.438776, expected=10325.000000
predicted=10624.630450, expected=9400.000000
predicted=9595.390996, expected=8585.000000
predicted=8713.243947, expected=8388.000000
predicted=8499.505778, expected=7047.000000
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/base/model.py:508: ConvergenceWarning: Maxi
  "Check mle_retvals", ConvergenceWarning)
```

```
predicted=7092.120122, expected=7041.000000
predicted=7084.686314, expected=6900.000000
predicted=6938.006485, expected=8140.000000
predicted=8220.091258, expected=9583.000000
predicted=9730.950465, expected=9311.000000
predicted=9436.675774, expected=9059.000000
predicted=9166.524373, expected=10012.000000
predicted=10165.041914, expected=10345.000000
predicted=10511.090474, expected=9249.000000
predicted=9348.186415, expected=7799.000000
predicted=7838.548094, expected=7877.000000
```

```
Test MAE: 731.838
Test RMSE: 956.257
```



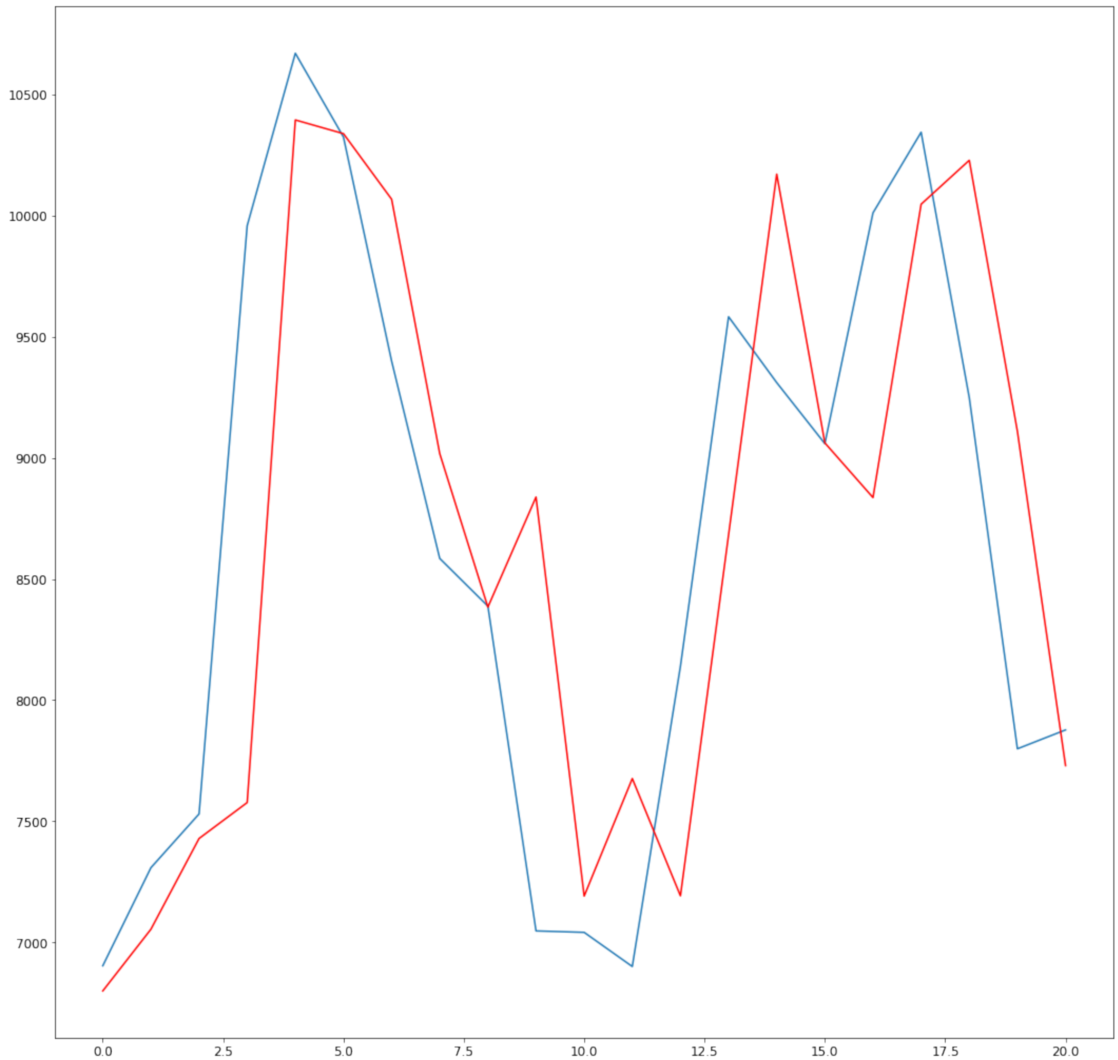
3.5 Autoregressive Integrated Moving Average (ARIMA)

ARIMA is an acronym that stands for AutoRegressive Integrated Moving Average. It is a class of model that captures a suite of different standard temporal structures in time series data. It explicitly caters to a suite of standard structures in time series data, and as such provides a simple yet powerful method for making skillful time series forecasts.

```
In [19]: history = [x for x in train]
         predictions = list()
         for t in range(len(test)):
             model = ARIMA(history, order=(5,1,0))
```

```
            model_fit = model.fit(disp=0)
            output = model_fit.forecast()
            yhat = output[0]
            predictions.append(yhat)
            obs = test[t]
            history.append(obs)
            print('predicted=%f, expected=%f' % (yhat, obs))
        ARIMA_RMSE = np.sqrt(mean_squared_error(test, predictions))
        ARIMA_MAE = mean_absolute_error(test, predictions)
        print('Test MAE: %.3f' % ARIMA_MAE)
        print('Test RMSE: %.3f' % ARIMA_RMSE)
        # plot
        plt.plot(test)
        plt.plot(predictions, color='red')
        plt.show()
```

```
predicted=6799.139167, expected=6903.000000
predicted=7053.893025, expected=7308.000000
predicted=7428.533470, expected=7530.000000
predicted=7577.024263, expected=9959.000000
predicted=10395.571460, expected=10671.000000
predicted=10339.012847, expected=10325.000000
predicted=10068.218664, expected=9400.000000
predicted=9017.867835, expected=8585.000000
predicted=8384.070188, expected=8388.000000
predicted=8838.612828, expected=7047.000000
predicted=7190.812583, expected=7041.000000
predicted=7676.169681, expected=6900.000000
predicted=7191.948232, expected=8140.000000
predicted=8682.306739, expected=9583.000000
predicted=10171.823821, expected=9311.000000
predicted=9062.093262, expected=9059.000000
predicted=8836.409541, expected=10012.000000
predicted=10047.601931, expected=10345.000000
predicted=10229.021111, expected=9249.000000
predicted=9109.585775, expected=7799.000000
predicted=7729.520400, expected=7877.000000
Test MAE: 646.533
Test RMSE: 897.763
```



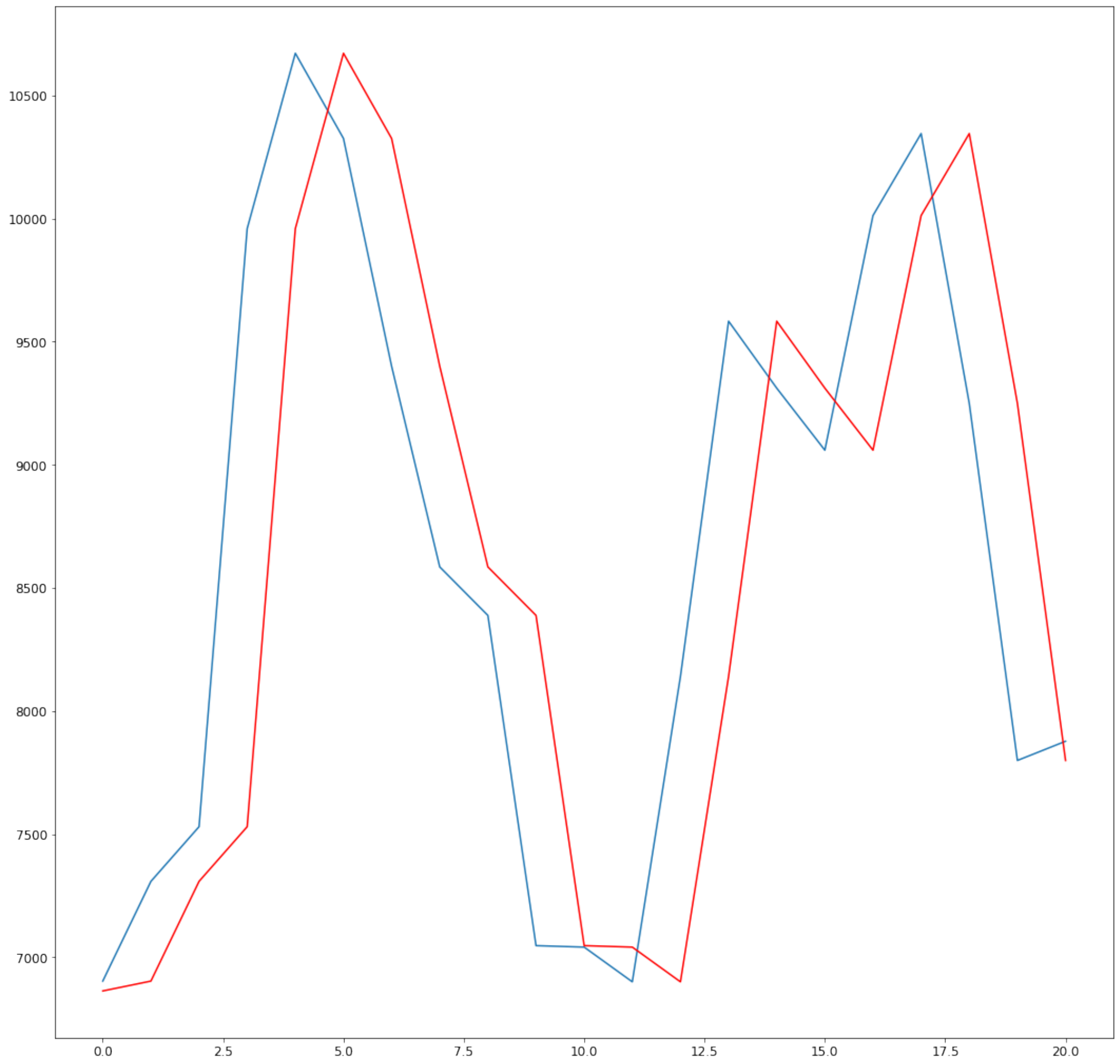
3.6 Exponential Smoothing

Exponential smoothing is a time series forecasting method for univariate data. It is similar in that a prediction is a weighted sum of past observations, but the model explicitly uses an exponentially decreasing weight for past observations.

```
In [20]: history = [x for x in train]
         predictions = list()
         for t in range(len(test)):
             model = ExponentialSmoothing(history)
             model_fit = model.fit()
             output = model_fit.predict()
             yhat = output[0]
             predictions.append(yhat)
             obs = test[t]
```

```python
            history.append(obs)
            print('predicted=%f, expected=%f' % (yhat, obs))
        ES_RMSE = np.sqrt(mean_squared_error(test, predictions))
        ES_MAE = mean_absolute_error(test, predictions)
        print('Test MAE: %.3f' % ES_MAE)
        print('Test RMSE: %.3f' % ES_RMSE)
        # plot
        plt.plot(test)
        plt.plot(predictions, color='red')
        plt.show()
```

```
predicted=6863.000000, expected=6903.000000
predicted=6903.000000, expected=7308.000000
predicted=7308.000000, expected=7530.000000
predicted=7530.000000, expected=9959.000000
predicted=9959.000000, expected=10671.000000
predicted=10671.000000, expected=10325.000000
predicted=10325.000000, expected=9400.000000
predicted=9400.000000, expected=8585.000000
predicted=8585.000000, expected=8388.000000
predicted=8388.000000, expected=7047.000000
predicted=7047.000000, expected=7041.000000
predicted=7041.000000, expected=6900.000000
predicted=6900.000000, expected=8140.000000
predicted=8140.000000, expected=9583.000000
predicted=9583.000000, expected=9311.000000
predicted=9311.000000, expected=9059.000000
predicted=9059.000000, expected=10012.000000
predicted=10012.000000, expected=10345.000000
predicted=10345.000000, expected=9249.000000
predicted=9249.000000, expected=7799.000000
predicted=7799.000000, expected=7877.000000
Test MAE: 699.810
Test RMSE: 931.070
```



3.7 Facebook Prophet

Prophet is designed for analyzing time series with daily observations that display patterns on different time scales. It also has advanced capabilities for modeling the effects of holidays on a time-series and implementing custom change points, but I will stick to the basic functions to get a model up and running.

```
In [21]: m = Prophet(yearly_seasonality=True)
         m.fit(df)
         future = m.make_future_dataframe(periods=365)
         # future.tail()
         forecast = m.predict(future)
         forecast.tail()
```

```
/usr/local/lib/python3.6/dist-packages/fbprophet/forecaster.py:250: FutureWarning: 'ds' is bot
Defaulting to column, but this will raise an ambiguity error in a future version
```

```
  df = df.sort_values('ds')
INFO:fbprophet:Disabling weekly seasonality. Run prophet with weekly_seasonality=True to overri
INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override
```

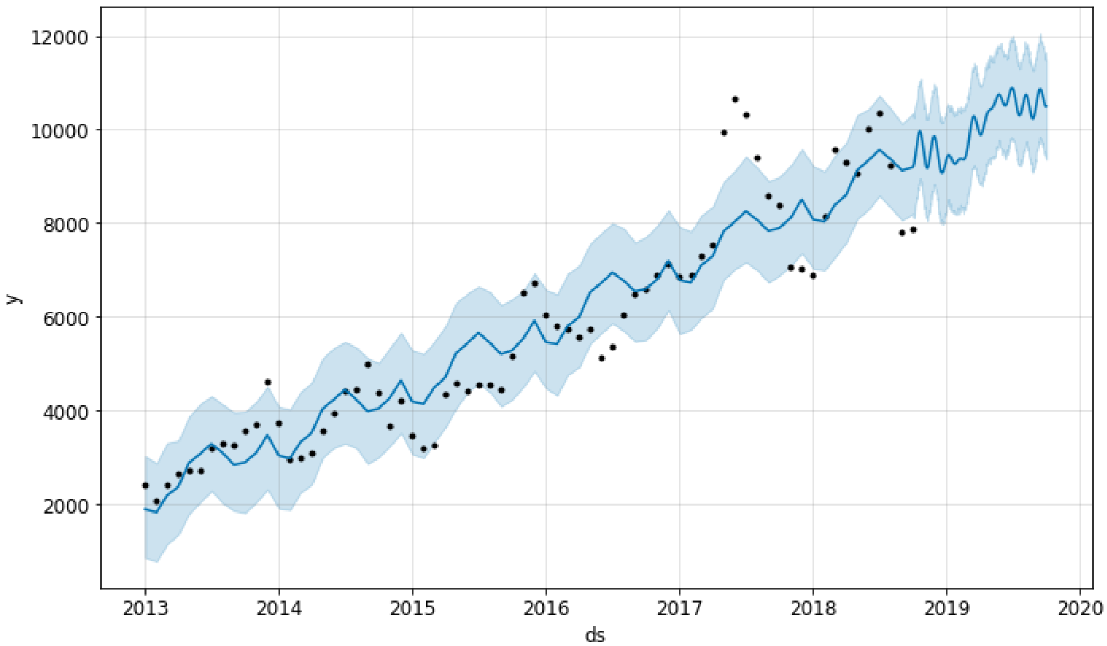
Out[21]:

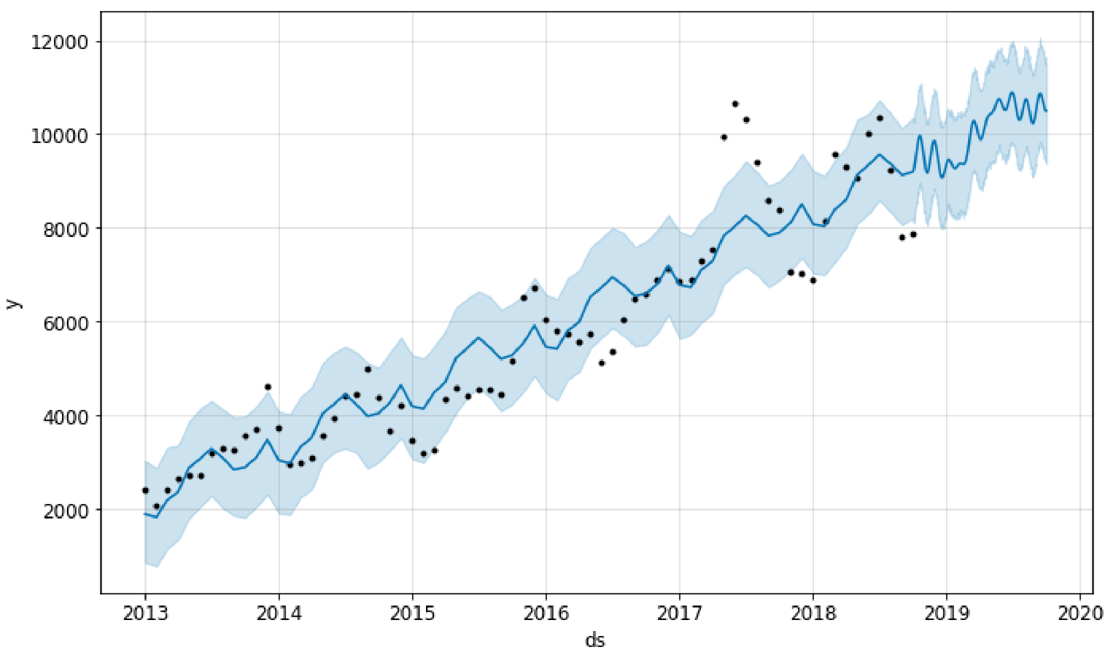
```
             ds         trend   yhat_lower    yhat_upper   trend_lower  \
430 2019-09-27  10676.337180  9441.395663  11620.627710  10664.179300
431 2019-09-28  10679.908100  9422.236640  11495.168577  10667.712194
432 2019-09-29  10683.479021  9395.497929  11648.367033  10671.245264
433 2019-09-30  10687.049942  9363.637160  11530.196921  10674.778334
434 2019-10-01  10690.620863  9362.880132  11461.517752  10678.311403

      trend_upper  additive_terms  additive_terms_lower  additive_terms_upper  \
430  10687.410947     -160.343192           -160.343192           -160.343192
431  10691.030840     -179.564110           -179.564110           -179.564110
432  10694.649114     -191.722860           -191.722860           -191.722860
433  10698.265724     -196.215243           -196.215243           -196.215243
434  10701.871611     -192.652113           -192.652113           -192.652113

         yearly  yearly_lower  yearly_upper  multiplicative_terms  \
430 -160.343192   -160.343192   -160.343192                   0.0
431 -179.564110   -179.564110   -179.564110                   0.0
432 -191.722860   -191.722860   -191.722860                   0.0
433 -196.215243   -196.215243   -196.215243                   0.0
434 -192.652113   -192.652113   -192.652113                   0.0

     multiplicative_terms_lower  multiplicative_terms_upper          yhat
430                         0.0                         0.0  10515.993988
431                         0.0                         0.0  10500.343990
432                         0.0                         0.0  10491.756161
433                         0.0                         0.0  10490.834698
434                         0.0                         0.0  10497.968749
```

In [22]:

```
m.plot(forecast)
```

Out[22]:



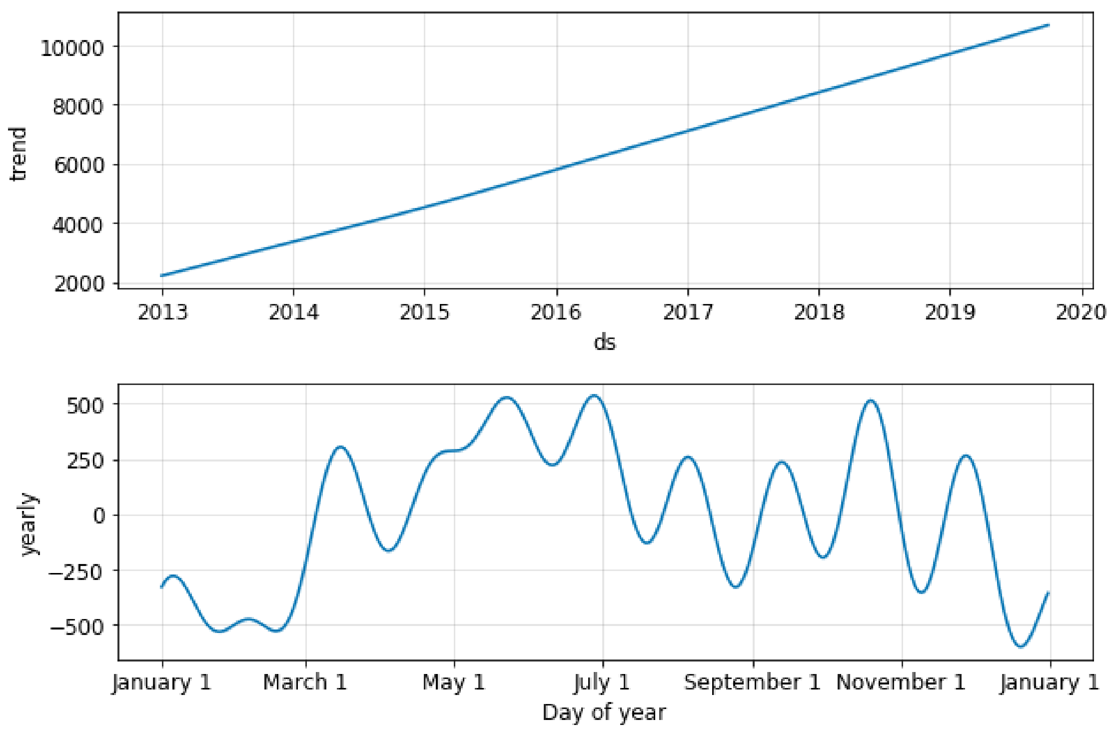


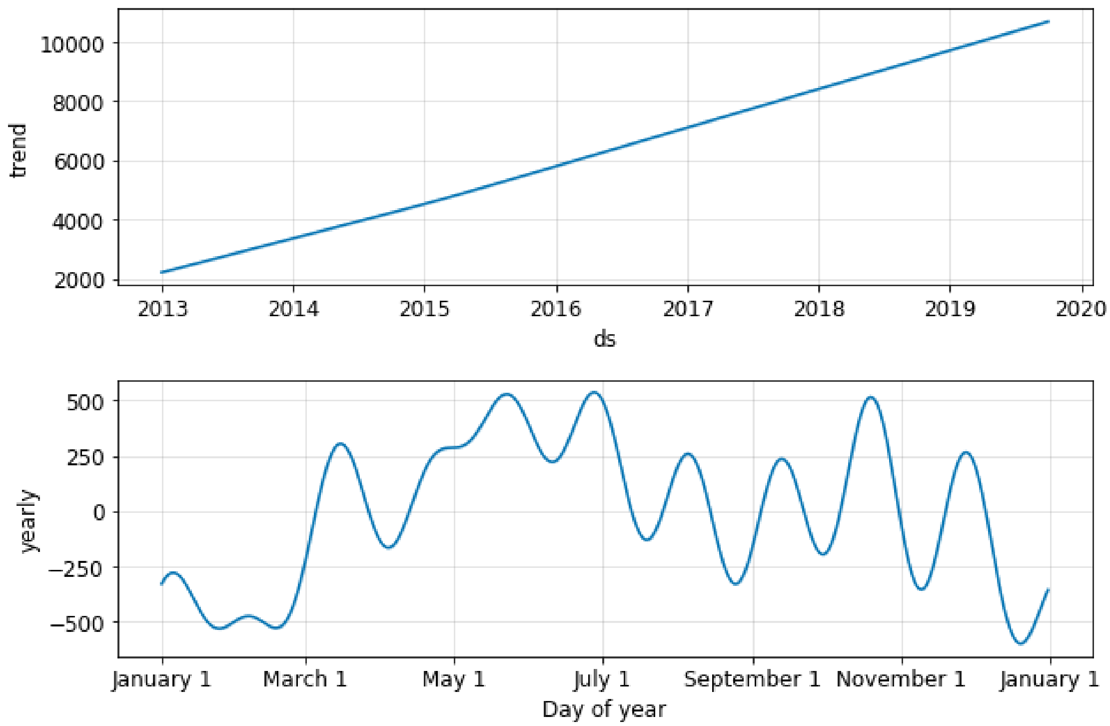
The black dots represent the actual values, the blue line indicates the forecasted values, and the light blue shaded region is the uncertainty (always a critical part of any prediction). The region of uncertainty increases the further out in the future the prediction is made because initial uncertainty propagates and grows over time. This is observed in weather forecasts which get less accurate the further out in time they are made.

In the following section, I will show that Prophet allows us to easily visualize the overall trend and the component patterns.

```
In [23]: m.plot_components(forecast)
```

```
Out[23]:
```






```
In [26]: forecast.index = pd.to_datetime(forecast.ds)
         forecast.sort_index(inplace=True)
         Prophet_MAE, Prophet_RMSE = evaluate_predictions(forecast['yhat'][:len(real)], real)
         print('Prophet  MAE: {:.4f}'.format(Prophet_MAE))
         print('Prophet RMSE: {:.4f}'.format(Prophet_RMSE))
```

```
Prophet  MAE: 655.4652
Prophet RMSE: 854.3138
```

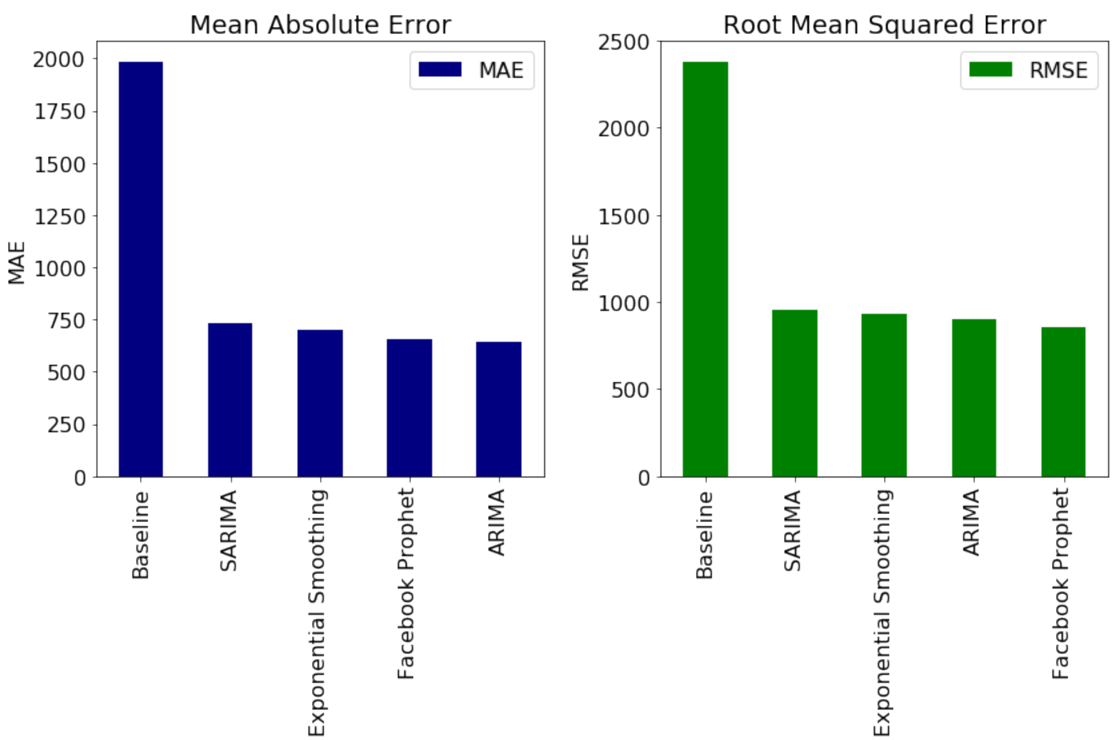
```
In [27]: # save forecasted 1-year results into a csv file
         results = forecast.loc[forecast['ds'] >= '2018-10-1', ['ds', 'yhat']]
         results.to_csv('forecasted_data_RAV4_sales.csv', sep=',', line_terminator='\n', encodi
```

3.8 Visual Comparison of Time Series Approaches

```
In [28]: # Evaluate forecasting approaches
         model_name_list = ['SARIMA', 'ARIMA', 'Exponential Smoothing', 'Facebook Prophet', 'Ba
         # Create a dataframe for results
         results = pd.DataFrame(columns=['MAE', 'RMSE'], index = model_name_list)

         # fill dataframe for results
         results.loc['Baseline', :] = [mb_MAE, mb_RMSE]
         results.loc['SARIMA', :] = [SARIMA_MAE, SARIMA_RMSE]
```

```
results.loc['ARIMA', :] = [ARIMA_MAE, ARIMA_RMSE]
results.loc['Exponential Smoothing', :] = [ES_MAE, ES_RMSE]
results.loc['Facebook Prophet', :] = [Prophet_MAE, Prophet_RMSE]
```

In [29]:
```
# Visualizing the evaluation results
figsize(12, 8)
matplotlib.rcParams['font.size'] = 16

# MAE plot
ax =  plt.subplot(1, 2, 1)
results.sort_values('MAE', ascending = False).plot.bar(y = 'MAE', color = 'navy', ax =
plt.title('Mean Absolute Error')
plt.ylabel('MAE')

# RMSE plot
ax = plt.subplot(1, 2, 2)
results.sort_values('RMSE', ascending = False).plot.bar(y = 'RMSE', color = 'g', ax =
plt.title('Root Mean Squared Error')
plt.ylabel('RMSE')
plt.tight_layout()
```



In [30]:
```
# show quantitative results
results
```

Out[30]:

	MAE	RMSE
SARIMA	731.838	956.257
ARIMA	646.533	897.763
Exponential Smoothing	699.81	931.07
Facebook Prophet	655.465	854.314
Baseline	1984.26	2379.02

4 Forecasting based on Two Different Trending Patterns

Based on simple observation from the historical dataset, we can easily see there are two different trending patterns: * one is from 1/1/2013 to 12/1/2016, * the other is from 1/1/2017 to 10/1/2018.

4.1 Separate DataSets based on Two Trending Patterns

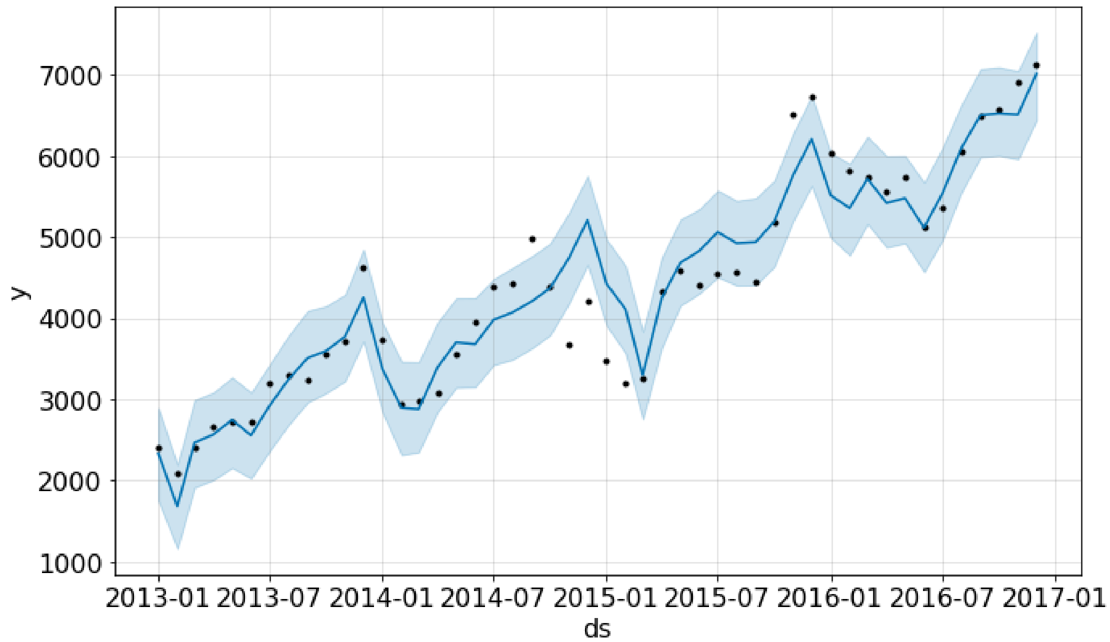
In [31]:
```
df1 = df[0:48]
df2 = df[48:]
```

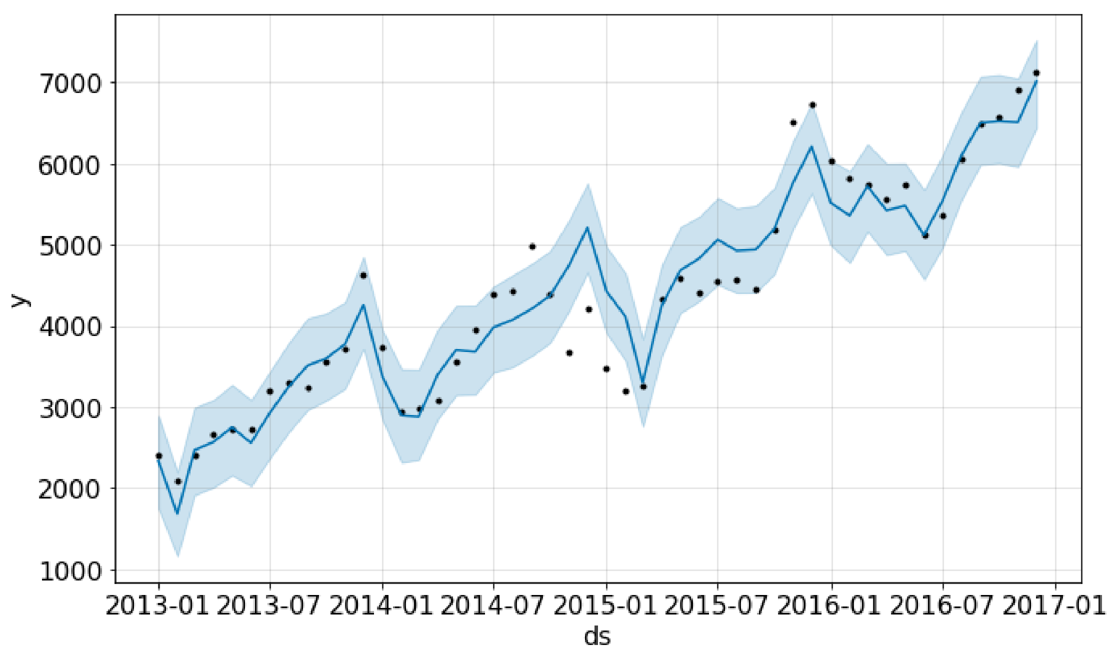
4.2 Forecasting for Period of 1/1/2013 - 12/1/2016

In [32]:
```
m1 = Prophet(yearly_seasonality=True)
m1.fit(df1)
future1 = m1.make_future_dataframe(periods=720)
# future.tail()
forecast1 = m1.predict(future1)
m1.plot(forecast1[:len(df1)])
```

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

Out[32]:






```
In [34]: real1 = df1['y']
         forecast1.index = pd.to_datetime(forecast1.ds)
         forecast1.sort_index(inplace=True)
         df_forecast1 = forecast1.loc[(forecast1['yhat'] >0), ['yhat']]
```

```
Prophet_MAE, Prophet_RMSE = evaluate_predictions(df_forecast1['yhat'][:len(real1)], re
print('Prophet  MAE: {:.4f}'.format(Prophet_MAE))
print('Prophet RMSE: {:.4f}'.format(Prophet_RMSE))
```

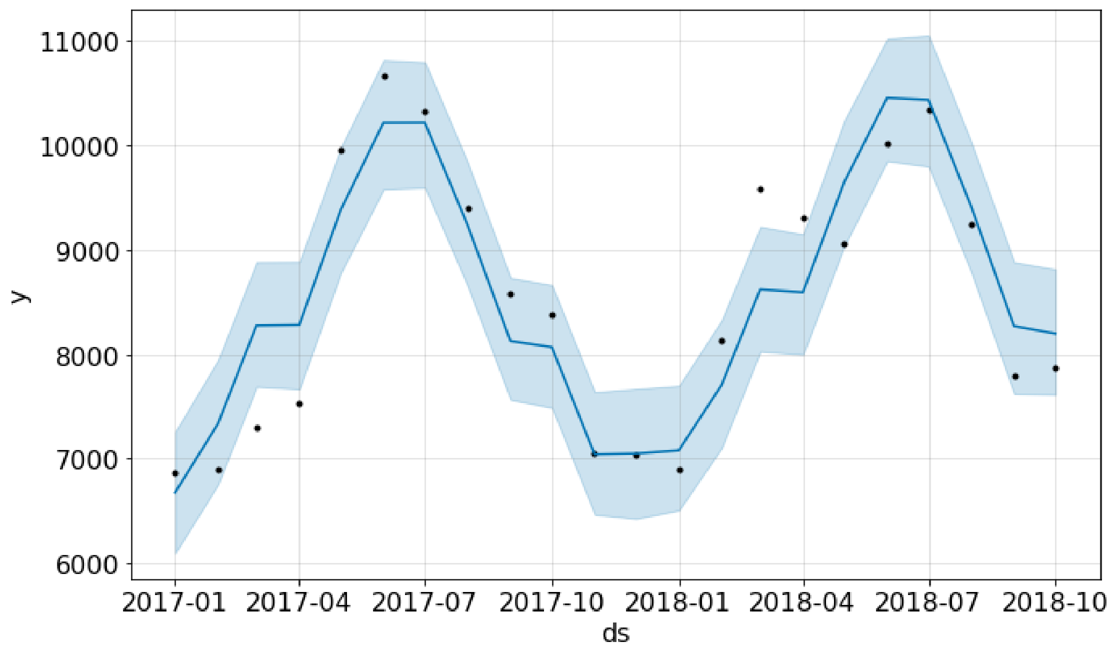
```
Prophet  MAE: 294.5200
Prophet RMSE: 409.7674
```

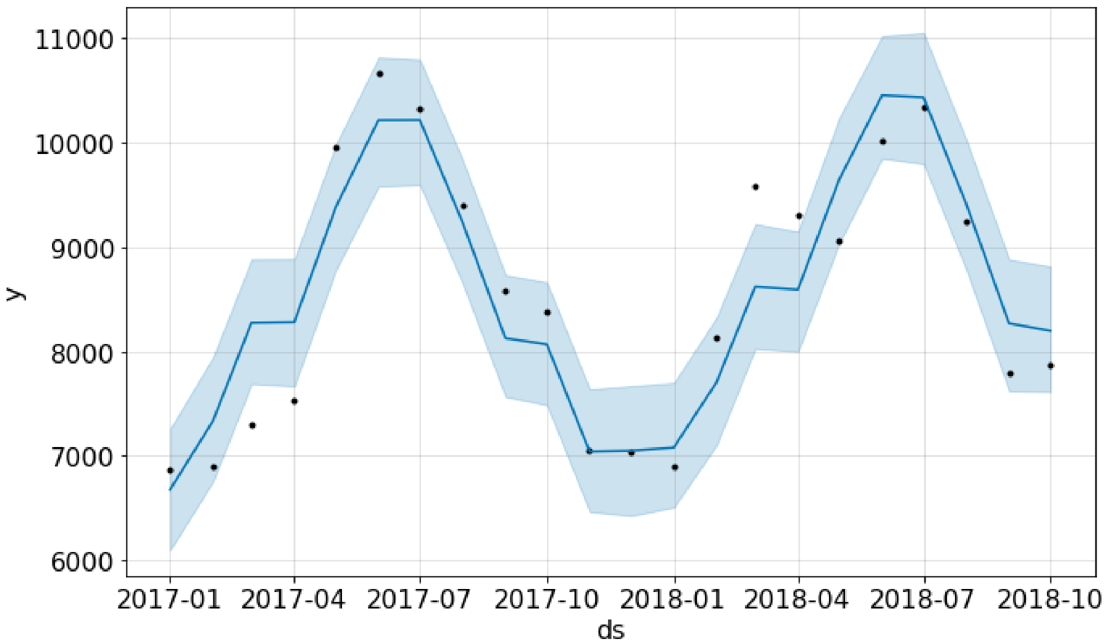
4.3 Forecasting for Period of 1/1/2017 - 10/1/2018

In [35]:
```
m2 = Prophet(yearly_seasonality=True)
m2.fit(df2)
future2 = m2.make_future_dataframe(periods=365)
# future.tail()
forecast2 = m2.predict(future2)
m2.plot(forecast2[:len(df2)])
```

```
INFO:fbprophet:Disabling weekly seasonality. Run prophet with weekly_seasonality=True to overri
INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override
INFO:fbprophet:n_changepoints greater than number of observations.Using 16.0.
```

Out[35]:






```
In [37]: # real2 = df2['y'].reset_index(drop=True)
         real2 = df2['y']
         forecast2.index = pd.to_datetime(forecast2.ds)
         forecast2.sort_index(inplace=True)
         df_forecast2 = forecast2.loc[(forecast2['yhat'] >0), ['yhat']]
         Prophet_MAE, Prophet_RMSE = evaluate_predictions(df_forecast2['yhat'][:len(real2)], r
         print('Prophet  MAE: {:.4f}'.format(Prophet_MAE))
         print('Prophet RMSE: {:.4f}'.format(Prophet_RMSE))
```

```
Prophet  MAE: 400.2010
Prophet RMSE: 485.1462
```

5 Conclusions

In this notebook I went through major steps to demonstrate how a time series forecasting project was implemented with a historical data of sales. A visual comparison of four different approaches was given. Additionally, we tried the forecasting based on two different trending data by slicing the data set into two periods. The final results presented the improved RMSEs.