Sri Eshwar
College of Engineering
An Autonomous Institution
Affiliated to Anna University, Chennai

SALES PREDICTION USING PROPHET

PROJECT REPORT

Submitted by

SARAVANA PERUMAL K : 722822104150

SUDHAN R : 722822104163

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

SRI ESHWAR COLLEGE OF ENGINEERING

(AN AUTONOMOUS INSTITUTION)

COIMBATORE – 641 202

JUNE - 2024

Sales Predictions with Prophet

Objectives

The objective for Sales Prediction with Facebook Prophet is to forecast future sales based on historical data while taking into account seasonality effects, demand, holidays, promotions, and competitions. This will enable businesses to make informed decisions regarding inventory management, resource allocation, and marketing strategies. By accurately predicting sales, businesses can optimize their operations, minimize costs, and maximize revenue. Additionally, the use of Facebook Prophet allows for the identification of patterns and trends in sales data, providing valuable insights for business planning and decision-making.

Methodology

1. Data Collection

This section of the code collects data from different branches of the same company. It is an essential step in the sales predictions process. The collected data will be used for further analysis and modeling using Facebook Prophet.

2. Data Preprocessing

Data preprocessing involves scaling the features using two different scalers: StandardScaler and MinMaxScaler. Scaling is a crucial step in data preprocessing that involves transforming the features to have a specific range or distribution. This is often done to ensure that the features contribute equally to the model and to improve the model's convergence during training.

1. **StandardScaler:** This scaler standardizes the features by removing the mean and scaling to unit variance.
2. **MinMaxScaler:** This scaler transforms the features by scaling them to a given range, typically between 0 and 1

3. Data Splitting

The data is split into training and testing sets using an 80-20 split. This ensures that 80% of the data is used for training the model, and 20% is reserved for testing its performance.

Libraries

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import datetime
```

Importing Datasets

1. Understanding the data

- **Id**: transaction ID (combination of Store and date)

- **Store**: unique store Id
- **Sales**: sales/day, this is the target variable
- **Customers**: number of customers on a given day
- **Open**: Boolean to say whether a store is open or closed (0 = closed, 1 = open)
- **Promo**: describes if store is running a promo on that day or not
- **StateHoliday**: indicate which state holiday (a = public holiday, b = Easter holiday, c = Christmas, 0 = None)
- **SchoolHoliday**: indicates if the (Store, Date) was affected by the closure of public schools

- **StoreType**: categorical variable to indicate type of store (a, b, c, d)

- **Assortment**: a = basic, b = extra, c = extended
- **CompetitionDistance** (meters): distance to closest competitor store
- **CompetitionOpenSince** [Month/Year]: date when competition was open
- **Promo2**: Promo2 is a continuing and consecutive promotion for some stores (0 = store is not participating, 1 = store is participating)
- **Promo2Since** [Year/Week]: date when store started participating in Promo2
- **PromoInterval**: describes the consecutive intervals Promo2 is started, naming the months promotion is started anew. E.g. "Feb,May,Aug,Nov" means each round starts in February, May, August, November of any given year for that store

2. Importing the dataset

In [2]:
```
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
/kaggle/input/rossman-datasets/store.csv
/kaggle/input/rossman-datasets/train.csv
```

In [3]:
```
# Uploading the store information data

# Data Source: https://www.kaggle.com/c/rossmann-store-sales/data

store_dataset = pd.read_csv("../input/rossman-datasets/store.csv")
store_dataset.head()
```

Out[3]:

	Store	StoreType	Assortment	CompetitionDistance	CompetitionOpenSinceMonth	CompetitionOpenSinceYear	Promo2	Pro
0	1	c	a	1270.0	9.0	2008.0	0	
1	2	a	a	570.0	11.0	2007.0	1	
2	3	a	a	14130.0	12.0	2006.0	1	
3	4	c	c	620.0	9.0	2009.0	0	
4	5	a	a	29910.0	4.0	2015.0	0	

In [4]:
```
# Importing the sales training data

# Data Source: https://www.kaggle.com/c/rossmann-store-sales/data

sales_dataset = pd.read_csv('../input/rossman-datasets/train.csv')
sales_dataset.head()
```

```
/opt/conda/lib/python3.7/site-packages/IPython/core/interactiveshell.py:3524: DtypeWarning: Columns (7) ha
ve mixed types.Specify dtype option on import or set low_memory=False.
  exec(code_obj, self.user_global_ns, self.user_ns)
```

Out[4]:

	Store	DayOfWeek	Date	Sales	Customers	Open	Promo	StateHoliday	SchoolHoliday
0	1	5	2015-07-31	5263	555	1	1	0	1
1	2	5	2015-07-31	6064	625	1	1	0	1
2	3	5	2015-07-31	8314	821	1	1	0	1
3	4	5	2015-07-31	13995	1498	1	1	0	1
4	5	5	2015-07-31	4822	559	1	1	0	1

Preparing and Exploring the Data

1. Sales Train dataset

In [5]:
```
sales_dataset.describe()

# Customers: Average of 633 customers
# Sales: Average of 5773 Euros per day
# Stores: 1017209 stores data in this dataset
```

Out[5]:

	Store	DayOfWeek	Sales	Customers	Open	Promo	SchoolHoliday
count	1.017209e+06	1.017209e+06	1.017209e+06	1.017209e+06	1.017209e+06	1.017209e+06	1.017209e+06
mean	5.584297e+02	3.998341e+00	5.773819e+03	6.331459e+02	8.301067e-01	3.815145e-01	1.786467e-01
std	3.219087e+02	1.997391e+00	3.849926e+03	4.644117e+02	3.755392e-01	4.857586e-01	3.830564e-01
min	1.000000e+00	1.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
25%	2.800000e+02	2.000000e+00	3.727000e+03	4.050000e+02	1.000000e+00	0.000000e+00	0.000000e+00
50%	5.580000e+02	4.000000e+00	5.744000e+03	6.090000e+02	1.000000e+00	0.000000e+00	0.000000e+00
75%	8.380000e+02	6.000000e+00	7.856000e+03	8.370000e+02	1.000000e+00	1.000000e+00	0.000000e+00
max	1.115000e+03	7.000000e+00	4.155100e+04	7.388000e+03	1.000000e+00	1.000000e+00	1.000000e+00

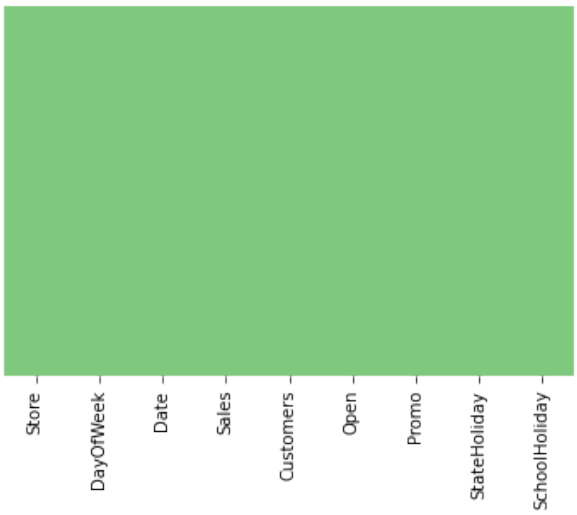
In [6]:
```
store_dataset.describe()

# Competitors distance on average: 5404 meters
# All columns will be used as features
```

Out[6]:

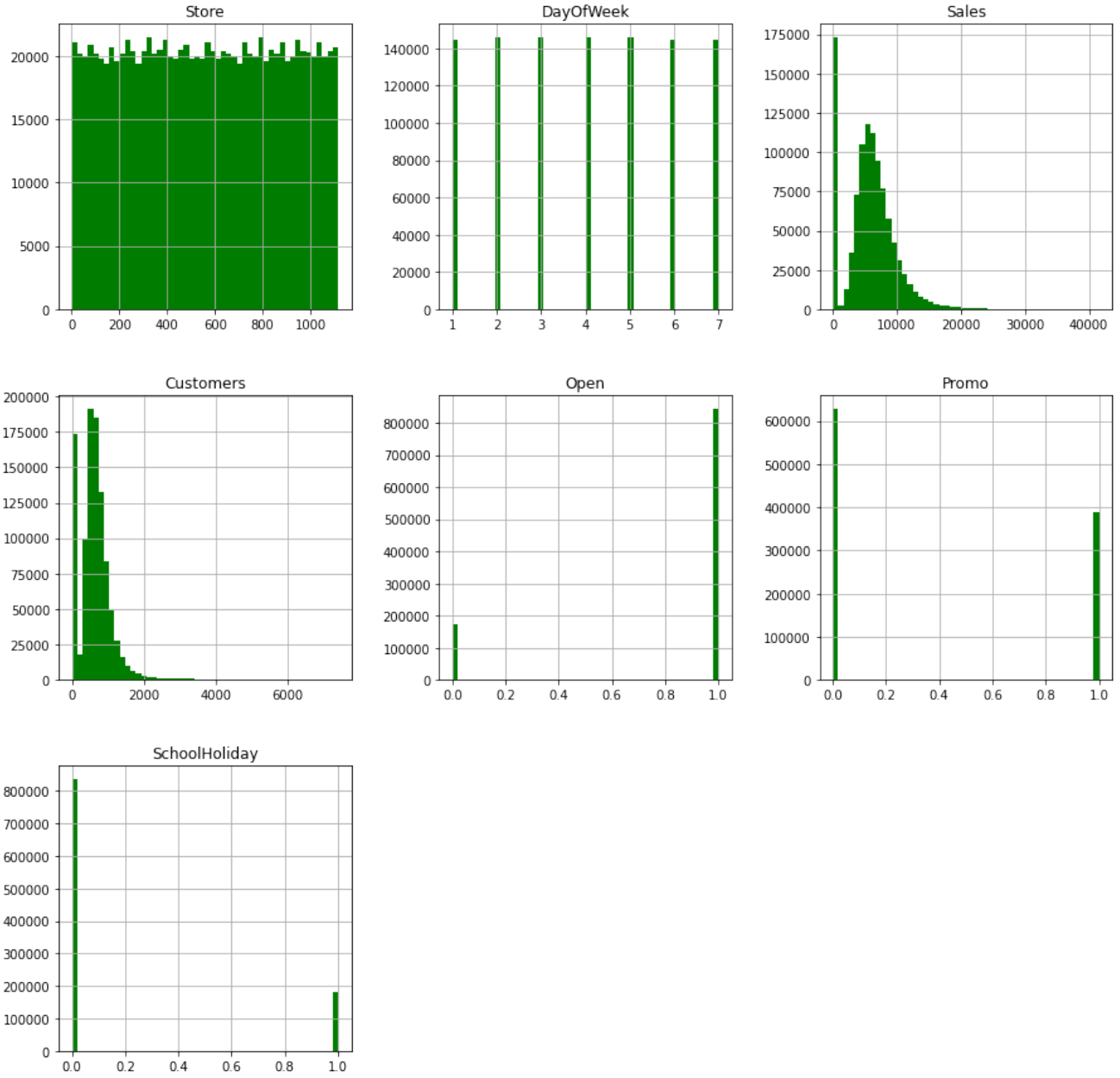
	Store	CompetitionDistance	CompetitionOpenSinceMonth	CompetitionOpenSinceYear	Promo2	Promo2SinceW
count	1115.00000	1112.000000	761.000000	761.000000	1115.000000	571.000(
mean	558.00000	5404.901079	7.224704	2008.668857	0.512108	23.595
std	322.01708	7663.174720	3.212348	6.195983	0.500078	14.141
min	1.00000	20.000000	1.000000	1900.000000	0.000000	1.000(
25%	279.50000	717.500000	4.000000	2006.000000	0.000000	13.000(
50%	558.00000	2325.000000	8.000000	2010.000000	1.000000	22.000(
75%	836.50000	6882.500000	10.000000	2013.000000	1.000000	37.000(
max	1115.00000	75860.000000	12.000000	2015.000000	1.000000	50.000(

In [7]:
```
# Checking for missing data

# There is no missing data on this dataframe

df_null = sales_dataset.isnull()
sns.heatmap(df_null, yticklabels=False, cbar=False, cmap = 'Accent')
```

Out[7]:
```
<AxesSubplot:>
```



In [8]:
```
# Plotting histograms for the Sales dataset

ax = sales_dataset.hist(bins = 50, figsize = (15,15), color = 'green')

# The data is distributed equaly across the week
# Sales decreases in school holidays
# Data is equaly distributed across the stores
```



In [9]:
```
# Stores are open 80% of the time

open = sales_dataset[sales_dataset['Open']==1]
closed = sales_dataset[sales_dataset['Open']==0]

print('Total = ',len(sales_dataset))
print('Number of stores that are open: ', len(open))
print('Percentage store that are open: ', 1.*len(open)/len(sales_dataset)*100, '%')

print('Number of stores that are closed: ', len(closed))
print('Percentage store that are closed: ', 1.*len(closed)/len(sales_dataset)*100, '%')
```

```
Total =  1017209
Number of stores that are open:  844392
Percentage store that are open:  83.01066939045958 %
Number of stores that are closed:  172817
Percentage store that are closed:  16.989330609540417 %
```

In [10]:
```
# Filtering the dataset to only the lines of Open stores

sales_dataset = sales_dataset[sales_dataset['Open']==1]
sales_dataset.drop(['Open'], axis = 1, inplace = True)
sales_dataset.shape
```

Out[10]:
```
(844392, 8)
```

In [11]:
```
# Actual average os sales is = 6955 Euros and Number os Customers = 762
sales_dataset.describe()
```

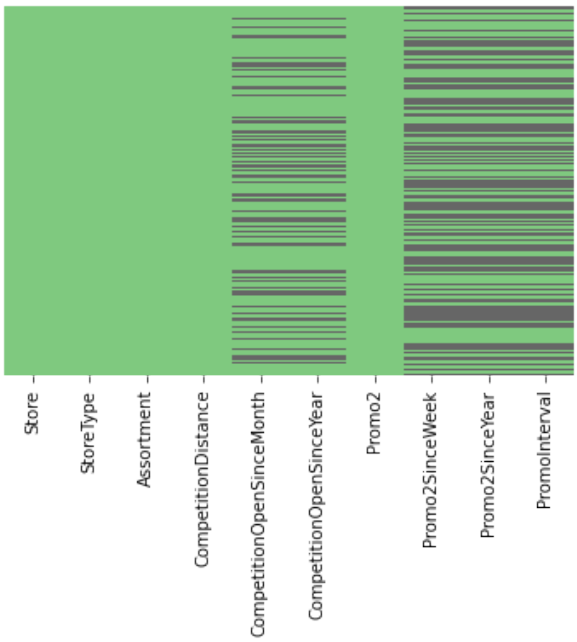
Out[11]:

	Store	DayOfWeek	Sales	Customers	Promo	SchoolHoliday
count	844392.000000	844392.000000	844392.000000	844392.000000	844392.000000	844392.000000
mean	558.422920	3.520361	6955.514291	762.728395	0.446352	0.193580
std	321.731914	1.723689	3104.214680	401.227674	0.497114	0.395103
min	1.000000	1.000000	0.000000	0.000000	0.000000	0.000000
25%	280.000000	2.000000	4859.000000	519.000000	0.000000	0.000000
50%	558.000000	3.000000	6369.000000	676.000000	0.000000	0.000000
75%	837.000000	5.000000	8360.000000	893.000000	1.000000	0.000000
max	1115.000000	7.000000	41551.000000	7388.000000	1.000000	1.000000

2. Stores dataset

In [12]:
```
# Checking for missing data

# There a lot of  missing data on this dataframe

df_null = store_dataset.isnull()
sns.heatmap(df_null, yticklabels=False, cbar=False, cmap = 'Accent')
```

Out[12]:
```
<AxesSubplot:>
```



In [13]:
```
# Treating the missing values

display(store_dataset[store_dataset['CompetitionDistance'].isnull()])
display(store_dataset[store_dataset['CompetitionOpenSinceMonth'].isnull()])
```

	Store	StoreType	Assortment	CompetitionDistance	CompetitionOpenSinceMonth	CompetitionOpenSinceYear	Promo2	P
290	291	d	a	NaN	NaN	NaN	0	
621	622	a	c	NaN	NaN	NaN	0	
878	879	d	a	NaN	NaN	NaN	1	

	Store	StoreType	Assortment	CompetitionDistance	CompetitionOpenSinceMonth	CompetitionOpenSinceYear	Promo2
11	12	a	c	1070.0	NaN	NaN	1
12	13	d	a	310.0	NaN	NaN	1
15	16	a	c	3270.0	NaN	NaN	0
18	19	a	c	3240.0	NaN	NaN	1
21	22	a	a	1040.0	NaN	NaN	1
...	...	...	...	...	...	...	...
1095	1096	a	c	1130.0	NaN	NaN	1
1099	1100	a	a	540.0	NaN	NaN	1
1112	1113	a	c	9260.0	NaN	NaN	0
1113	1114	a	c	870.0	NaN	NaN	0
1114	1115	d	c	5350.0	NaN	NaN	1

354 rows × 10 columns

In [14]:
```
display(store_dataset[store_dataset['Promo2'].isnull()])
display(store_dataset[store_dataset['Promo2SinceWeek'].isnull()])
```

	Store	StoreType	Assortment	CompetitionDistance	CompetitionOpenSinceMonth	CompetitionOpenSinceYear	Promo2	Prom

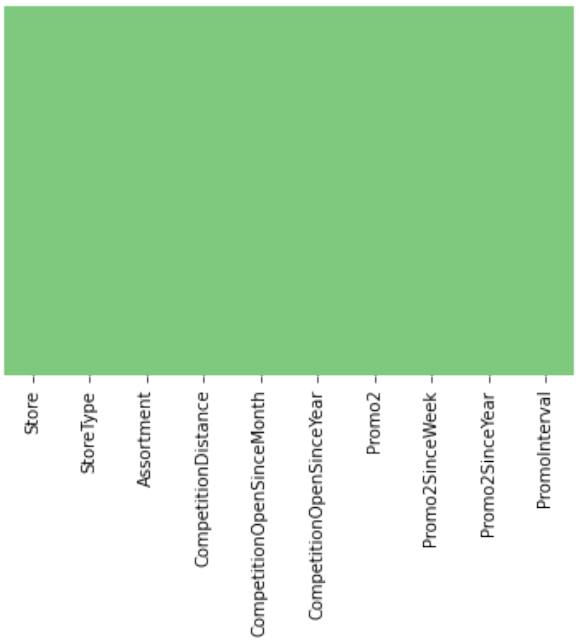
	Store	StoreType	Assortment	CompetitionDistance	CompetitionOpenSinceMonth	CompetitionOpenSinceYear	Promo2
0	1	c	a	1270.0	9.0	2008.0	0
3	4	c	c	620.0	9.0	2009.0	0
4	5	a	a	29910.0	4.0	2015.0	0
5	6	a	a	310.0	12.0	2013.0	0
6	7	a	c	24000.0	4.0	2013.0	0
...	...	...	...	...	...	...	...
1107	1108	a	a	540.0	4.0	2004.0	0
1109	1110	c	c	900.0	9.0	2010.0	0
1111	1112	c	c	1880.0	4.0	2006.0	0
1112	1113	a	c	9260.0	NaN	NaN	0
1113	1114	a	c	870.0	NaN	NaN	0

544 rows × 10 columns

In [15]:
```
# Deleting columns that are not relevant (toomuch null values):
del_col = ['Promo2SinceWeek',  'Promo2SinceYear',
           'PromoInterval', 'CompetitionOpenSinceMonth',
           'CompetitionOpenSinceYear']

for n in del_col:
  store_dataset[n].fillna(0, inplace = True)
```

In [16]:
```
# Checking how is now
df_null = store_dataset.isnull()
sns.heatmap(df_null, yticklabels=False, cbar=False, cmap = 'Accent')
```

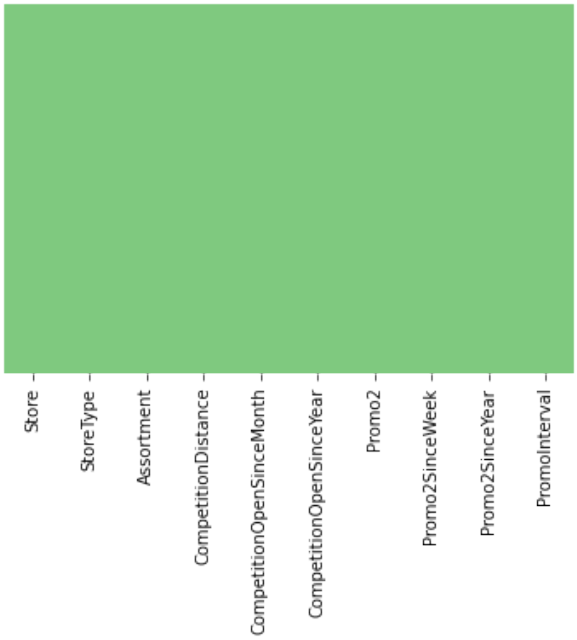
Out[16]:
```
<AxesSubplot:>
```



In [17]:
```
# Treating the misisng data of CompetitionDistance

store_dataset['CompetitionDistance'].fillna(store_dataset['CompetitionDistance'].mean(), inplace = True)
```

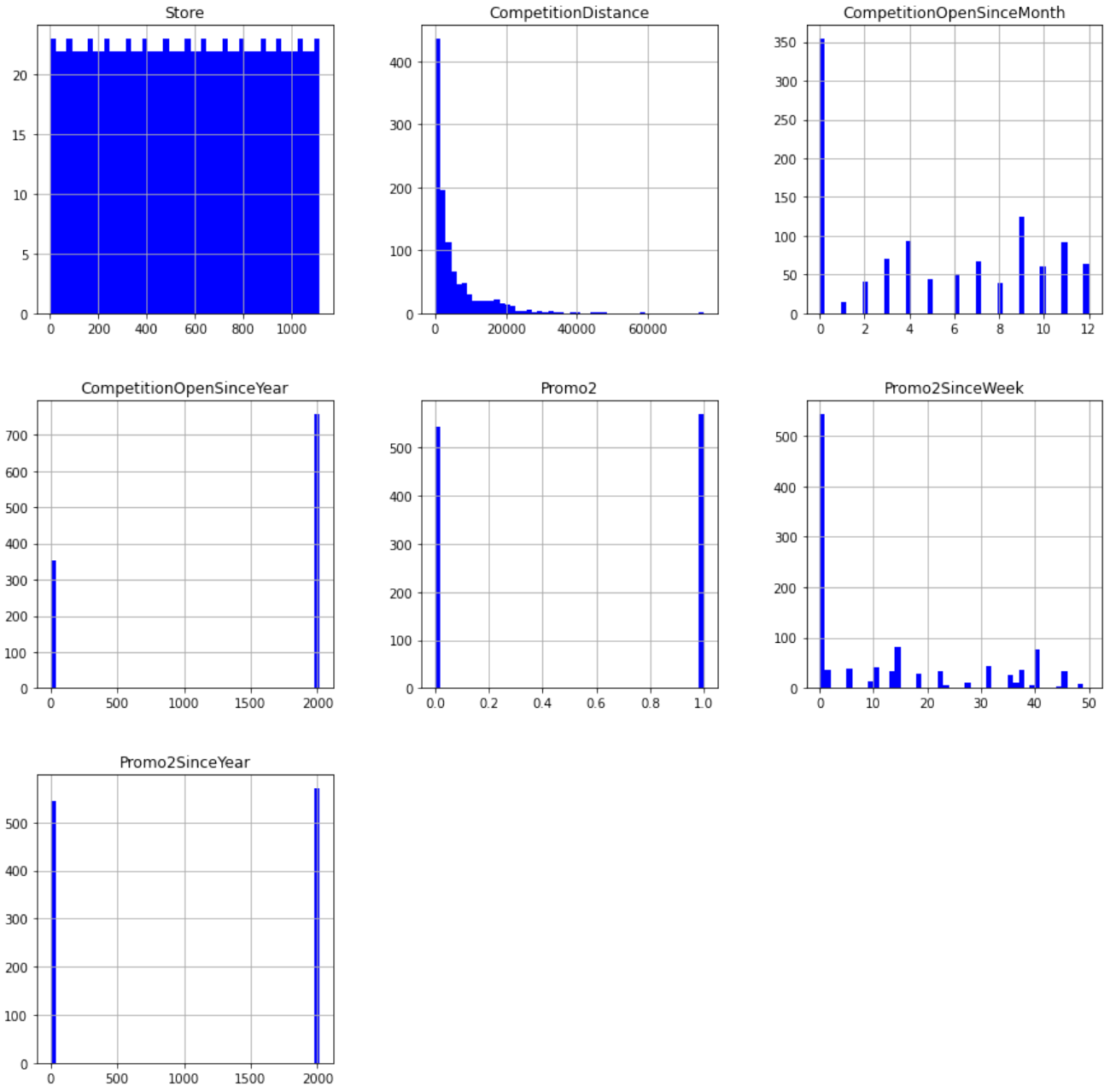
In [18]:
```
sns.heatmap(store_dataset.isnull(), yticklabels=False, cbar=False, cmap = 'Accent')
```

Out[18]:
```
<AxesSubplot:>
```



In [19]:
```
# Plotting Histograms

store_dataset.hist(bins = 50, figsize = (15,15), color = 'blue')
```

Out[19]:
```
array([[<AxesSubplot:title={'center':'Store'}>,
        <AxesSubplot:title={'center':'CompetitionDistance'}>,
        <AxesSubplot:title={'center':'CompetitionOpenSinceMonth'}>],
       [<AxesSubplot:title={'center':'CompetitionOpenSinceYear'}>,
        <AxesSubplot:title={'center':'Promo2'}>,
        <AxesSubplot:title={'center':'Promo2SinceWeek'}>],
       [<AxesSubplot:title={'center':'Promo2SinceYear'}>, <AxesSubplot:>,
        <AxesSubplot:>]], dtype=object)
```



3. Merging the two datasets

Store column is the best to use to merge the two dataframes

In [20]:
```
sales_dataset.head(3)
```

Out[20]:

	Store	DayOfWeek	Date	Sales	Customers	Promo	StateHoliday	SchoolHoliday
0	1	5	2015-07-31	5263	555	1	0	1
1	2	5	2015-07-31	6064	625	1	0	1
2	3	5	2015-07-31	8314	821	1	0	1

In [21]:
```
store_dataset.head(3)
```

Out[21]:

	Store	StoreType	Assortment	CompetitionDistance	CompetitionOpenSinceMonth	CompetitionOpenSinceYear	Promo2	Pro
0	1	c	a	1270.0	9.0	2008.0	0	
1	2	a	a	570.0	11.0	2007.0	1	
2	3	a	a	14130.0	12.0	2006.0	1	

In [22]:
```
# Merging

all_dataset = pd.merge(sales_dataset, store_dataset, how = 'inner', on = 'Store')
all_dataset.head(3)
```

Out[22]:

	Store	DayOfWeek	Date	Sales	Customers	Promo	StateHoliday	SchoolHoliday	StoreType	Assortment	CompetitionDista
0	1	5	2015-07-31	5263	555	1	0	1	c	a	12
1	1	4	2015-07-30	5020	546	1	0	1	c	a	12
2	1	3	2015-07-29	4782	523	1	0	1	c	a	12

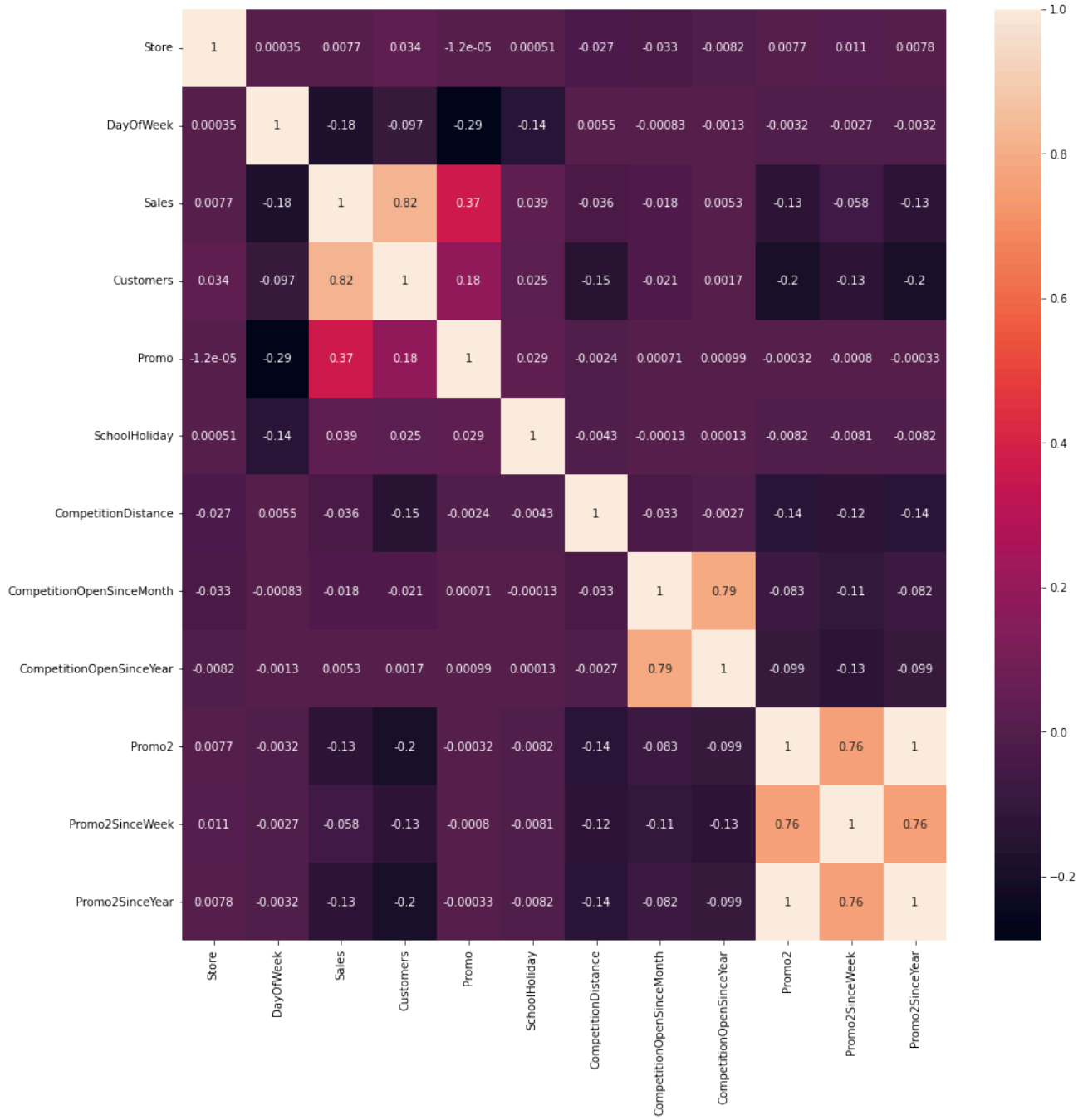
4. Checking the correlations between the features

In [23]:
```
correlations = all_dataset.corr()['Sales'].sort_values()
correlations
```

Out[23]:
```
DayOfWeek                  -0.178736
Promo2SinceYear            -0.127621
Promo2                     -0.127596
Promo2SinceWeek            -0.058476
CompetitionDistance        -0.036343
CompetitionOpenSinceMonth  -0.018370
CompetitionOpenSinceYear    0.005266
Store                       0.007710
SchoolHoliday               0.038617
Promo                       0.368145
Customers                   0.823597
Sales                       1.000000
Name: Sales, dtype: float64
```

In [24]:
```
#Visualizing with a heatmap

correlations = all_dataset.corr()
g, ax = plt.subplots(figsize = (15,15))
sns.heatmap(correlations, annot = True)
```

Out[24]:
```
<AxesSubplot:>
```



Data Visualization

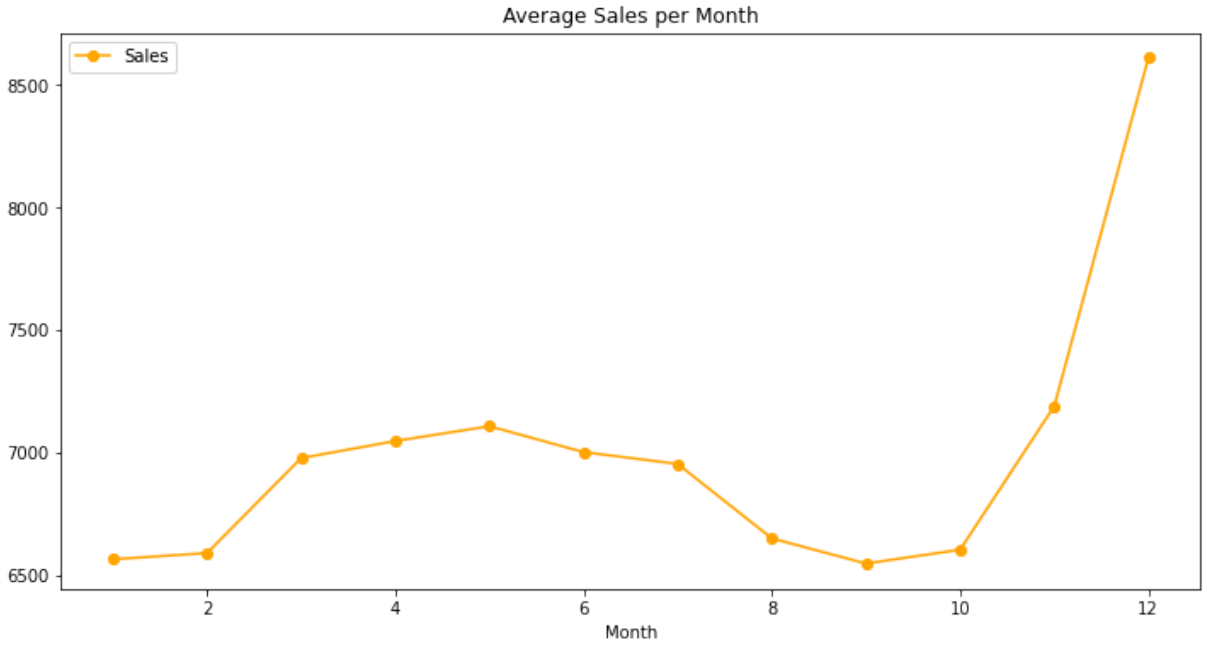
In [25]:
```
# Treating the date

all_dataset['Year'] = pd.DatetimeIndex(all_dataset['Date']).year
all_dataset['Month'] = pd.DatetimeIndex(all_dataset['Date']).month
all_dataset['Day'] = pd.DatetimeIndex(all_dataset['Date']).day
all_dataset.head(3)
```

Out[25]:

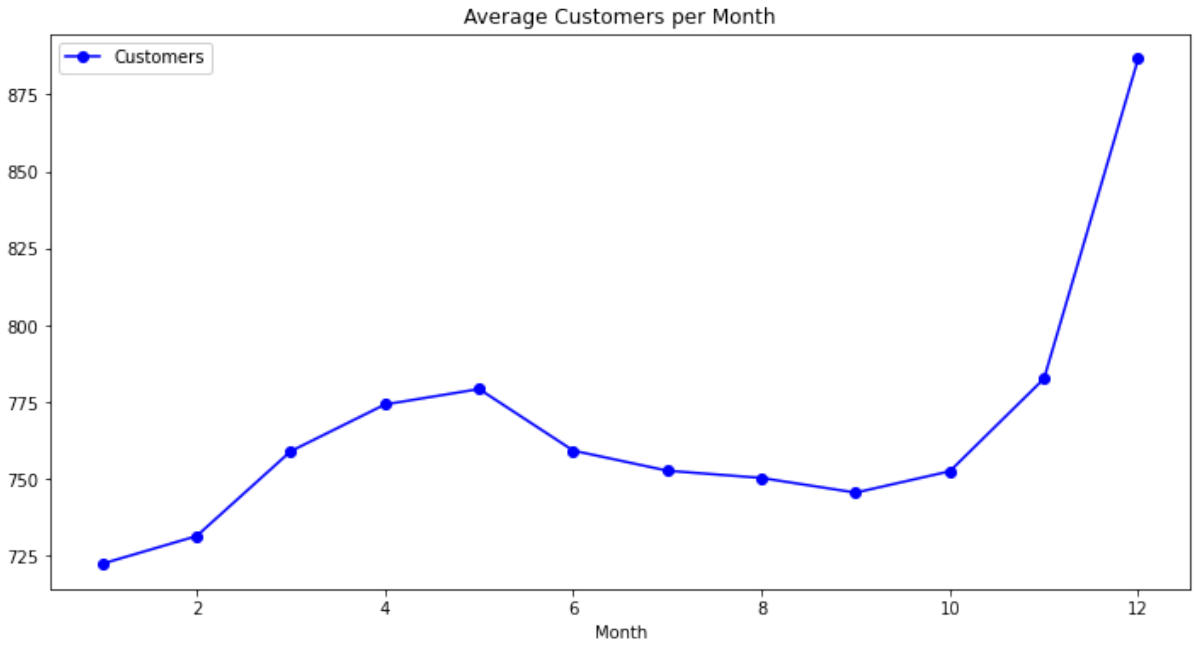
	Store	DayOfWeek	Date	Sales	Customers	Promo	StateHoliday	SchoolHoliday	StoreType	Assortment	CompetitionDista
0	1	5	2015-07-31	5263	555	1	0	1	c	a	12
1	1	4	2015-07-30	5020	546	1	0	1	c	a	12
2	1	3	2015-07-29	4782	523	1	0	1	c	a	12

In [26]:
```
# Average sales and number of customer per Month

ax = all_dataset.groupby('Month')[['Sales']].mean().plot(figsize = (12,6), marker = 'o', color = 'orange')
ax.set_title('Average Sales per Month')

plt.figure()

ax = all_dataset.groupby('Month')[['Customers']].mean().plot(figsize = (12,6), marker = 'o', color = 'blue'
ax.set_title('Average Customers per Month')

# Sales and customers has pick around november and december
```

Out[26]:
```
Text(0.5, 1.0, 'Average Customers per Month')
```

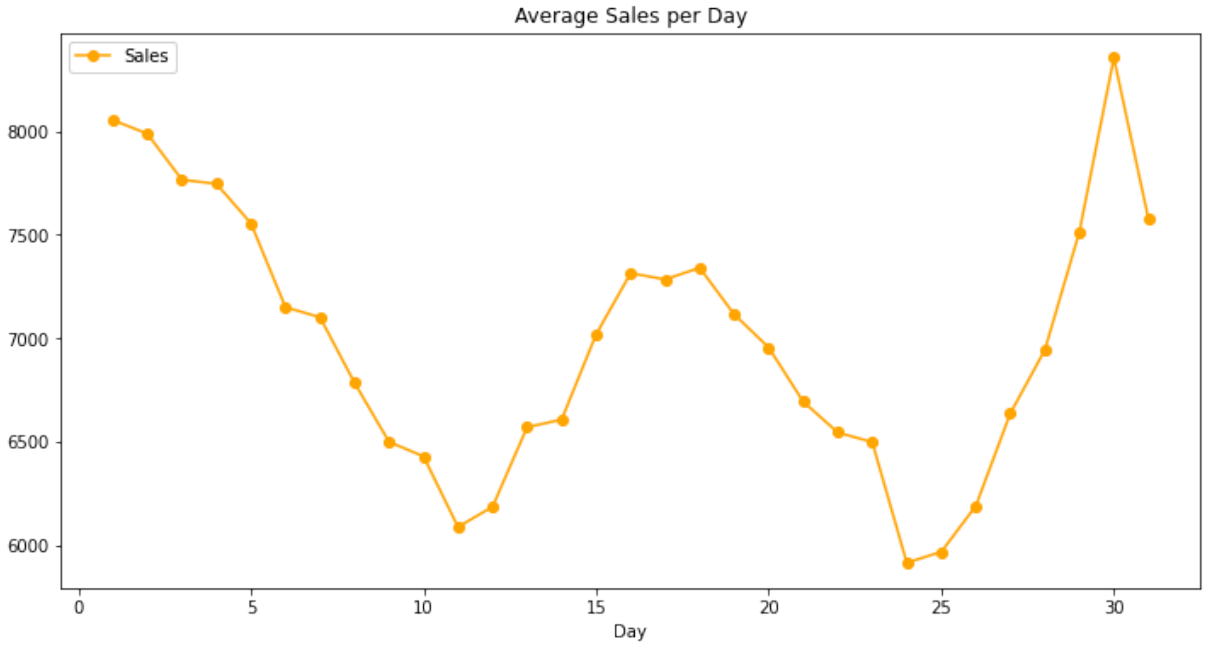



```
<Figure size 432x288 with 0 Axes>
```

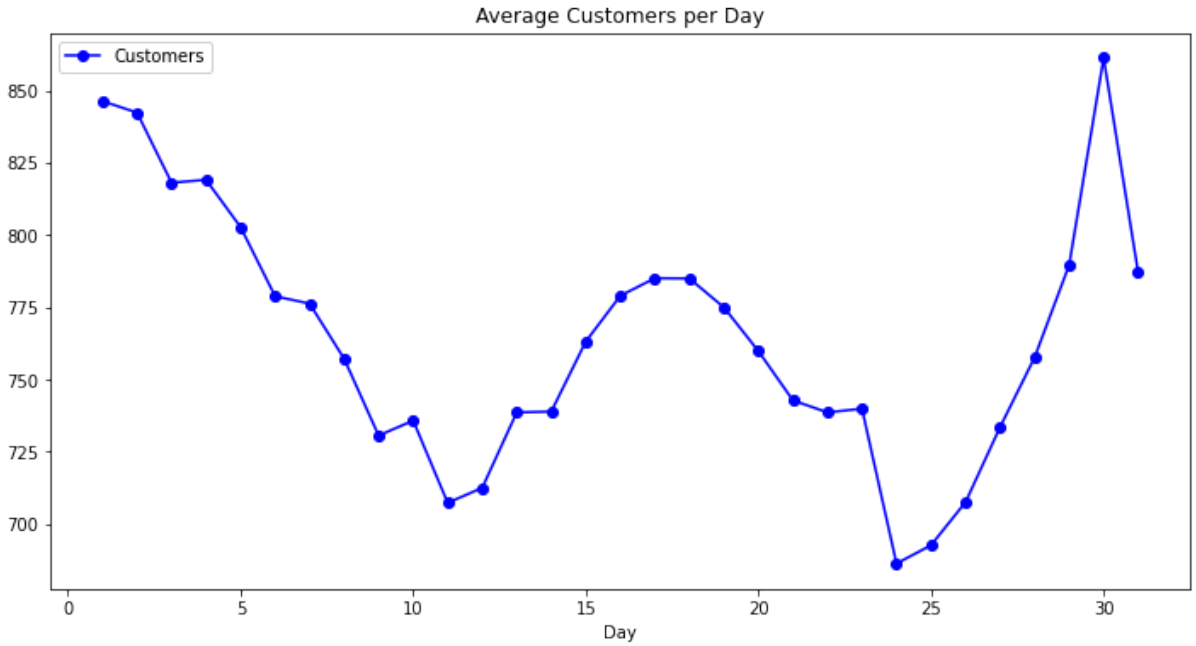


In [27]:
```
# Checking the average sales and customers per day

ax = all_dataset.groupby('Day')[['Sales']].mean().plot(figsize = (12,6), marker = 'o', color = 'orange')
ax.set_title('Average Sales per Day')

plt.figure()

ax = all_dataset.groupby('Day')[['Customers']].mean().plot(figsize = (12,6), marker = 'o', color = 'blue')
ax.set_title('Average Customers per Day')
```

Out[27]:
```
Text(0.5, 1.0, 'Average Customers per Day')
```

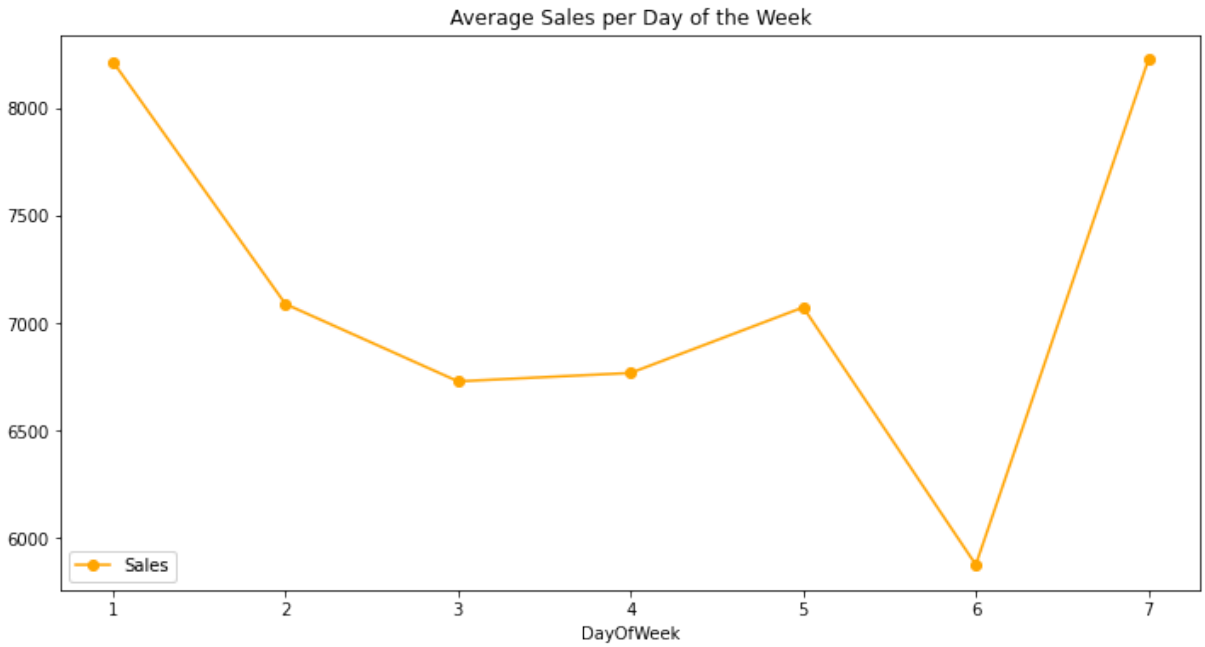



```
<Figure size 432x288 with 0 Axes>
```

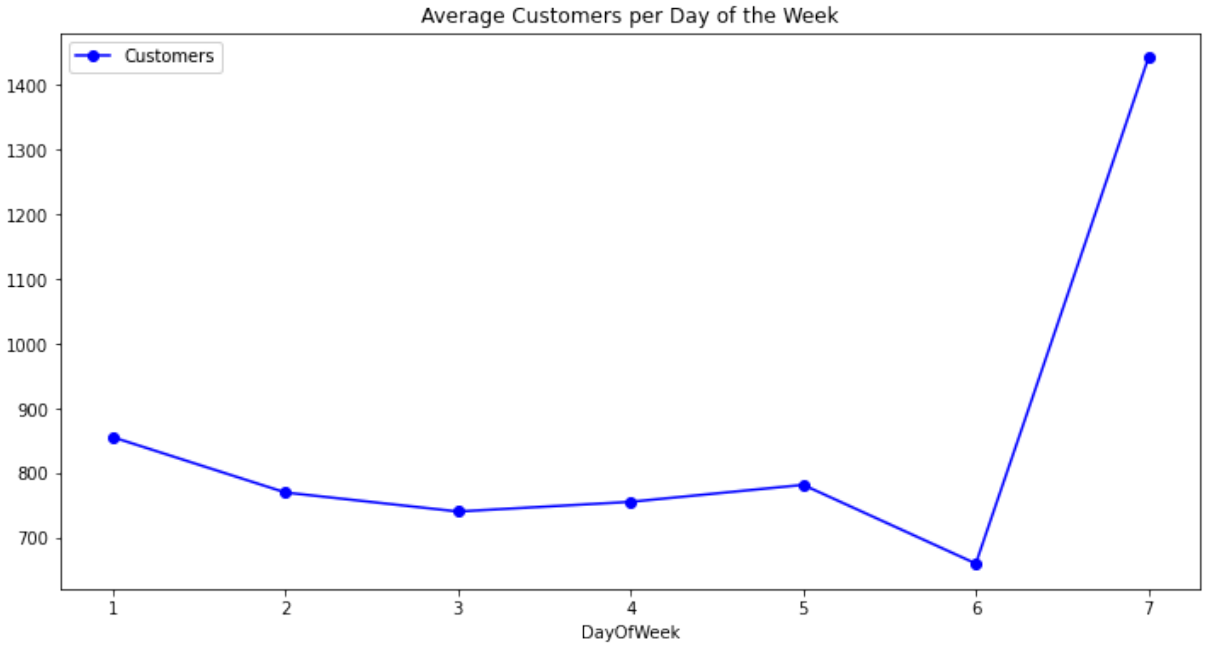


In [28]:
```
# Checking the average sales and customers per day of the week

ax = all_dataset.groupby('DayOfWeek')[['Sales']].mean().plot(figsize = (12,6), marker = 'o', color = 'orang
ax.set_title('Average Sales per Day of the Week')

plt.figure()

ax = all_dataset.groupby('DayOfWeek')[['Customers']].mean().plot(figsize = (12,6), marker = 'o', color = 'b
ax.set_title('Average Customers per Day of the Week')

# Pick of sales and customers on sunday
```

Out[28]:
```
Text(0.5, 1.0, 'Average Customers per Day of the Week')
```



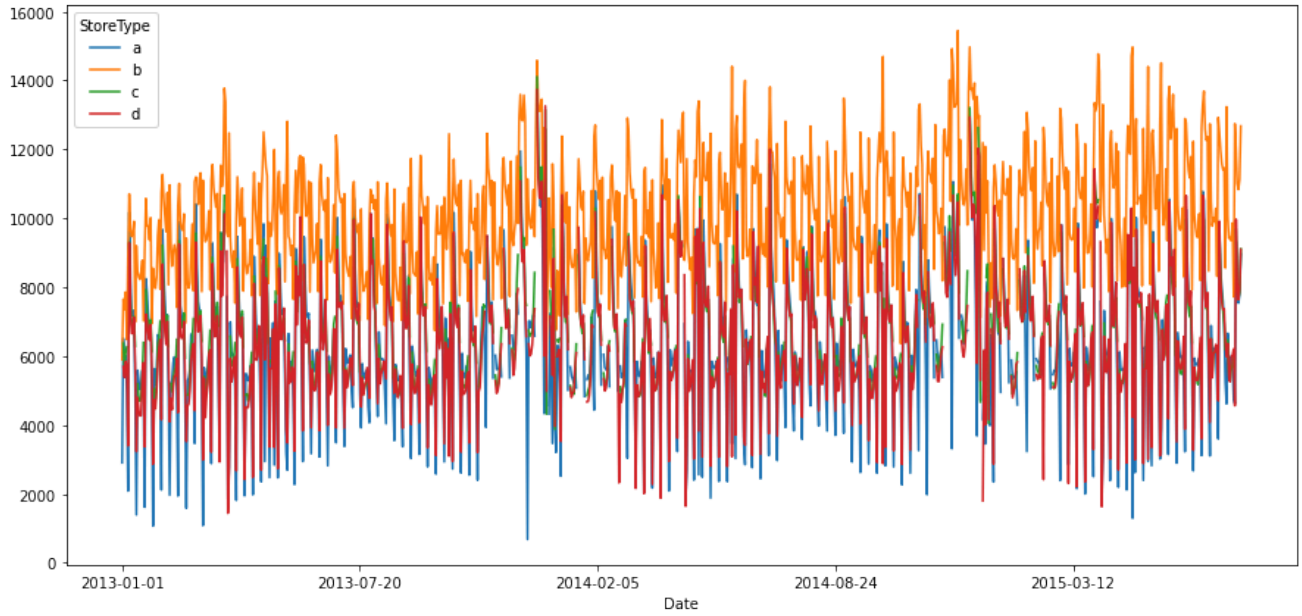

```
<Figure size 432x288 with 0 Axes>
```



Conclusion is that there is an influence of seasonality

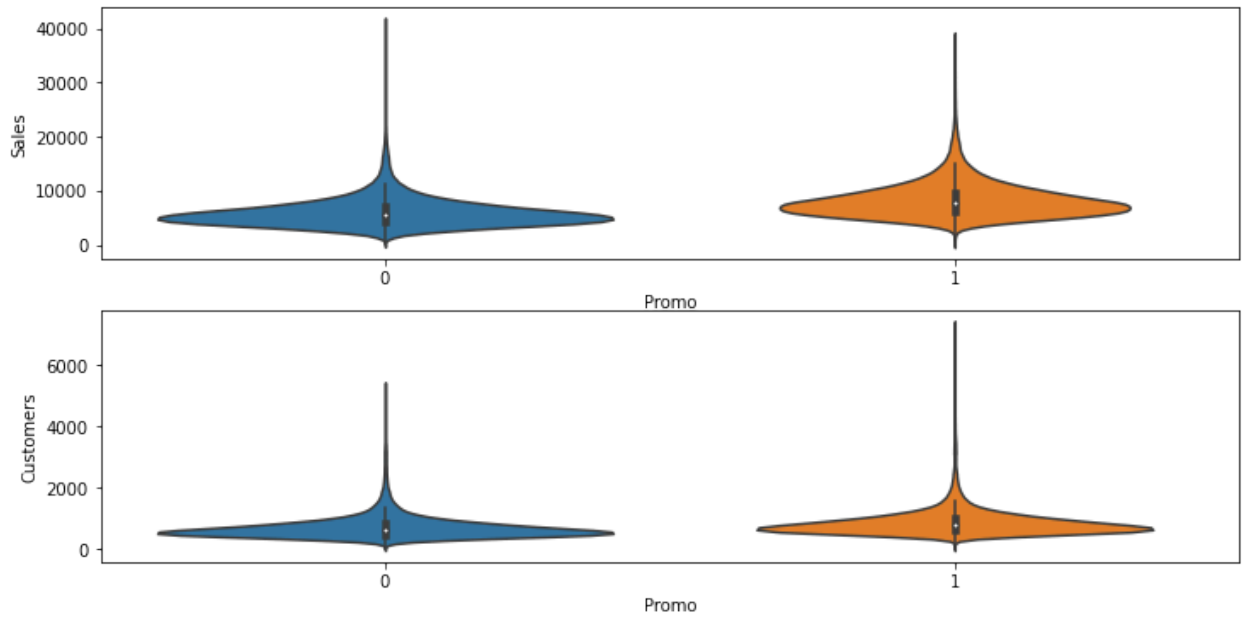
In [29]:
```
fig, ax = plt.subplots(figsize = (15,7))

all_dataset.groupby(['Date', 'StoreType']).mean()['Sales'].unstack().plot(ax = ax)

# Store b has high amount of sales
```

Out[29]:
```
<AxesSubplot:xlabel='Date'>
```



In [30]:
```
# checking if the promos has any effect on sales

plt.figure(figsize = (12, 6))
plt.subplot(211)
sns.violinplot(x = 'Promo', y = 'Sales', data = all_dataset)
plt.subplot(212)
sns.violinplot(x = 'Promo', y = 'Customers', data = all_dataset)

# Sales and customers are higher with promo
```

Out[30]:
```
<AxesSubplot:xlabel='Promo', ylabel='Customers'>
```



Sales Prediction using Prophet

Facebook Prophet is highly used for a time series and the data has strong seasonal effects and several seasons of historical data.

1. Making predictions (time series only)

In [1]:
```
# Install fbprophet

! pip install prophet
```

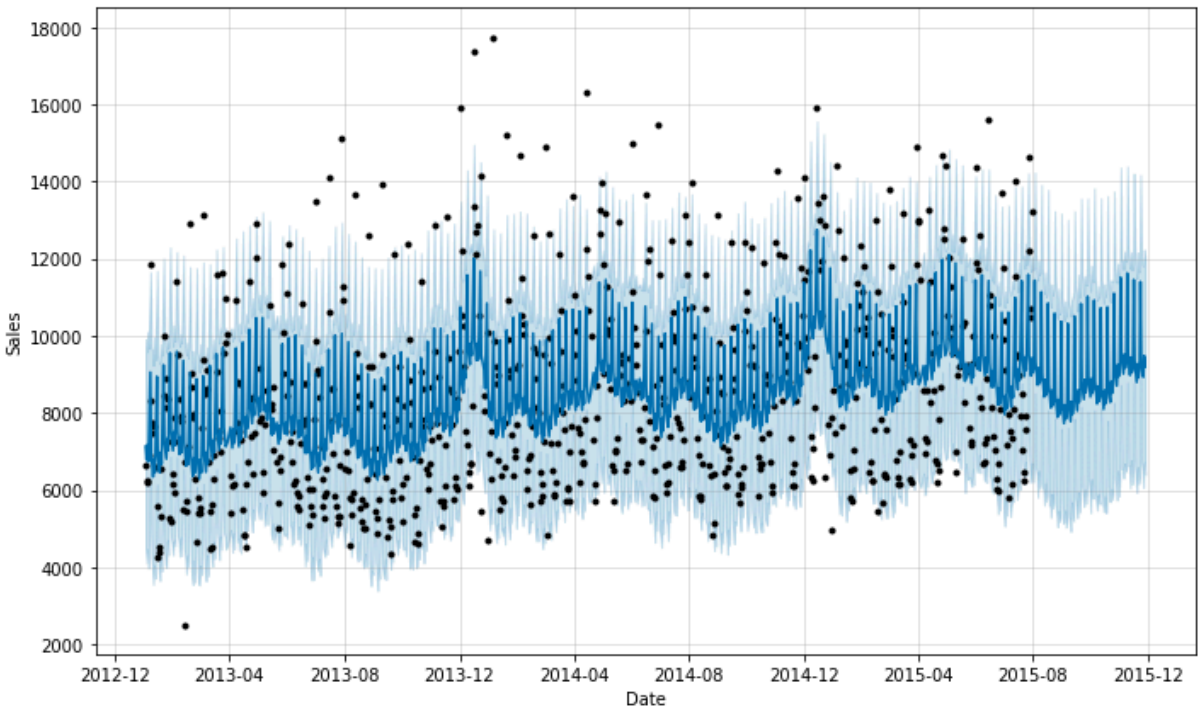
```
Requirement already satisfied: prophet in c:\users\sudha\anaconda3\envs\geoprocessing\lib\site-packages
(1.1.4)
Requirement already satisfied: cmdstanpy>=1.0.4 in c:\users\sudha\anaconda3\envs\geoprocessing\lib\site-pa
ckages (from prophet) (1.1.0)
Requirement already satisfied: numpy>=1.15.4 in c:\users\sudha\anaconda3\envs\geoprocessing\lib\site-packa
ges (from prophet) (1.25.0)
Requirement already satisfied: matplotlib>=2.0.0 in c:\users\sudha\anaconda3\envs\geoprocessing\lib\site-p
ackages (from prophet) (3.7.1)
Requirement already satisfied: pandas>=1.0.4 in c:\users\sudha\anaconda3\envs\geoprocessing\lib\site-packa
ges (from prophet) (1.5.3)
Requirement already satisfied: LunarCalendar>=0.0.9 in c:\users\sudha\anaconda3\envs\geoprocessing\lib\sit
e-packages (from prophet) (0.0.9)
Requirement already satisfied: convertdate>=2.1.2 in c:\users\sudha\anaconda3\envs\geoprocessing\lib\site-
packages (from prophet) (2.3.2)
Requirement already satisfied: holidays>=0.25 in c:\users\sudha\anaconda3\envs\geoprocessing\lib\site-pack
ages (from prophet) (0.48)
Requirement already satisfied: python-dateutil>=2.8.0 in c:\users\sudha\anaconda3\envs\geoprocessing\lib\s
ite-packages (from prophet) (2.8.2)
Requirement already satisfied: tqdm>=4.36.1 in c:\users\sudha\anaconda3\envs\geoprocessing\lib\site-packag
es (from prophet) (4.66.1)
Requirement already satisfied: importlib-resources in c:\users\sudha\anaconda3\envs\geoprocessing\lib\site
-packages (from prophet) (6.1.1)
Requirement already satisfied: pytz>=2014.10 in c:\users\sudha\anaconda3\envs\geoprocessing\lib\site-packa
ges (from convertdate>=2.1.2->prophet) (2022.7)
Requirement already satisfied: pymeeus<=1,>=0.3.13 in c:\users\sudha\anaconda3\envs\geoprocessing\lib\site
-packages (from convertdate>=2.1.2->prophet) (0.5.11)
Requirement already satisfied: ephem>=3.7.5.3 in c:\users\sudha\anaconda3\envs\geoprocessing\lib\site-pack
ages (from LunarCalendar>=0.0.9->prophet) (4.1.2)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\sudha\anaconda3\envs\geoprocessing\lib\site-pa
ckages (from matplotlib>=2.0.0->prophet) (1.0.5)
Requirement already satisfied: cycler>=0.10 in c:\users\sudha\anaconda3\envs\geoprocessing\lib\site-packag
es (from matplotlib>=2.0.0->prophet) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\sudha\anaconda3\envs\geoprocessing\lib\site-p
ackages (from matplotlib>=2.0.0->prophet) (4.25.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\sudha\anaconda3\envs\geoprocessing\lib\site-p
ackages (from matplotlib>=2.0.0->prophet) (1.4.4)
Requirement already satisfied: packaging>=20.0 in c:\users\sudha\anaconda3\envs\geoprocessing\lib\site-pac
kages (from matplotlib>=2.0.0->prophet) (23.0)
Requirement already satisfied: pillow>=6.2.0 in c:\users\sudha\anaconda3\envs\geoprocessing\lib\site-packa
ges (from matplotlib>=2.0.0->prophet) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\sudha\anaconda3\envs\geoprocessing\lib\site-pa
ckages (from matplotlib>=2.0.0->prophet) (3.0.9)
Requirement already satisfied: six>=1.5 in c:\users\sudha\anaconda3\envs\geoprocessing\lib\site-packages
(from python-dateutil>=2.8.0->prophet) (1.16.0)
Requirement already satisfied: colorama in c:\users\sudha\anaconda3\envs\geoprocessing\lib\site-packages
(from tqdm>=4.36.1->prophet) (0.4.6)
```

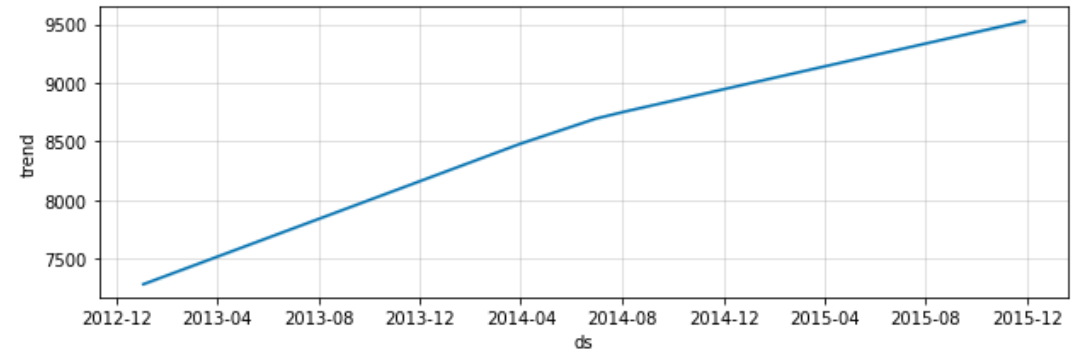
In [32]:
```python
# Import fbprophet

from prophet import Prophet
```

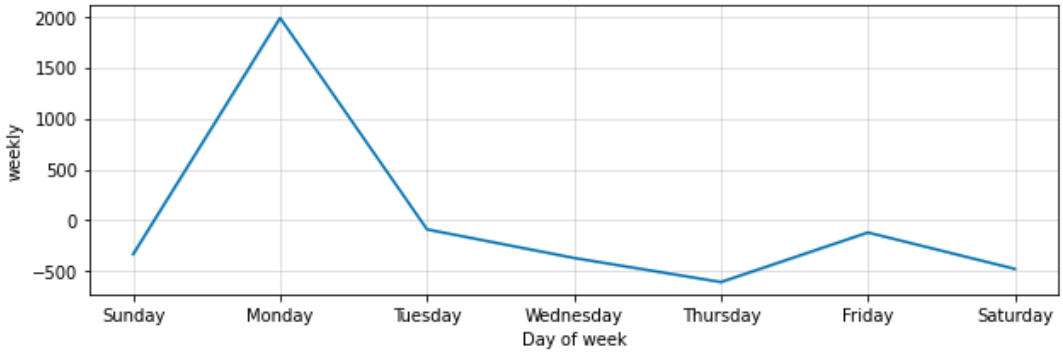
In [33]:
```python
# Predictions

# Creating a df with sales data from one specific store and period:

selected_store = 250
periods = 120  #future days

sales_store = all_dataset[all_dataset['Store'] == selected_store]

# For the fbprophet, df must have columns "ds" and "y" with the dates and values respectively.

sales_store = sales_store[['Date', 'Sales']].rename(columns = {'Date' : 'ds', 'Sales' : 'y'})
sales_store = sales_store.sort_values('ds')

# Training the model and predicting for this specific store

model = Prophet()
model.fit(sales_store)
future_df = model.make_future_dataframe(periods = periods)
predictions = model.predict(future_df)
```

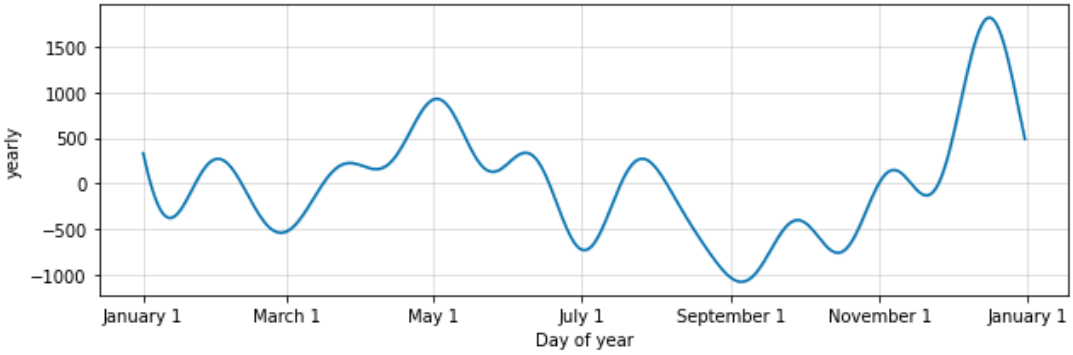
```
Initial log joint probability = -13.7002
    Iter      log prob        ||dx||      ||grad||       alpha      alpha0  # evals  Notes
      99       1255.34   9.76275e-05       65.9723           1           1      125
    Iter      log prob        ||dx||      ||grad||       alpha      alpha0  # evals  Notes
     198        1255.4   7.01188e-08       84.2365           1           1      246
Optimization terminated normally:
  Convergence detected: relative gradient magnitude is below tolerance
```

In [34]:
```
# Plotting the result

ax = model.plot(predictions, xlabel = 'Date', ylabel = 'Sales')
ax2 = model.plot_components(predictions)
```









2. Making predictions considering the holidays

In [35]:
```
# There are two types of holydays, school holidays and state holydays

school_holidays = all_dataset[all_dataset['SchoolHoliday'] == 1].loc[:, 'Date'].values

state_holidays = all_dataset[(all_dataset['StateHoliday'] == 'a')|(all_dataset['StateHoliday'] == 'b')|(all

school_holidays = pd.DataFrame({'ds' : pd.to_datetime(school_holidays),
                                'holiday': 'school_holiday'})

state_holidays = pd.DataFrame({'ds' : pd.to_datetime(state_holidays),
                                'holiday': 'state_holiday'})

display(school_holidays.shape)
display(state_holidays.shape)
```

```
(163457, 2)
(910, 2)
```

In [36]:
```
# Concatenate both holidays

holidays_all = pd.concat((state_holidays,school_holidays))
holidays_all.head(3)
```

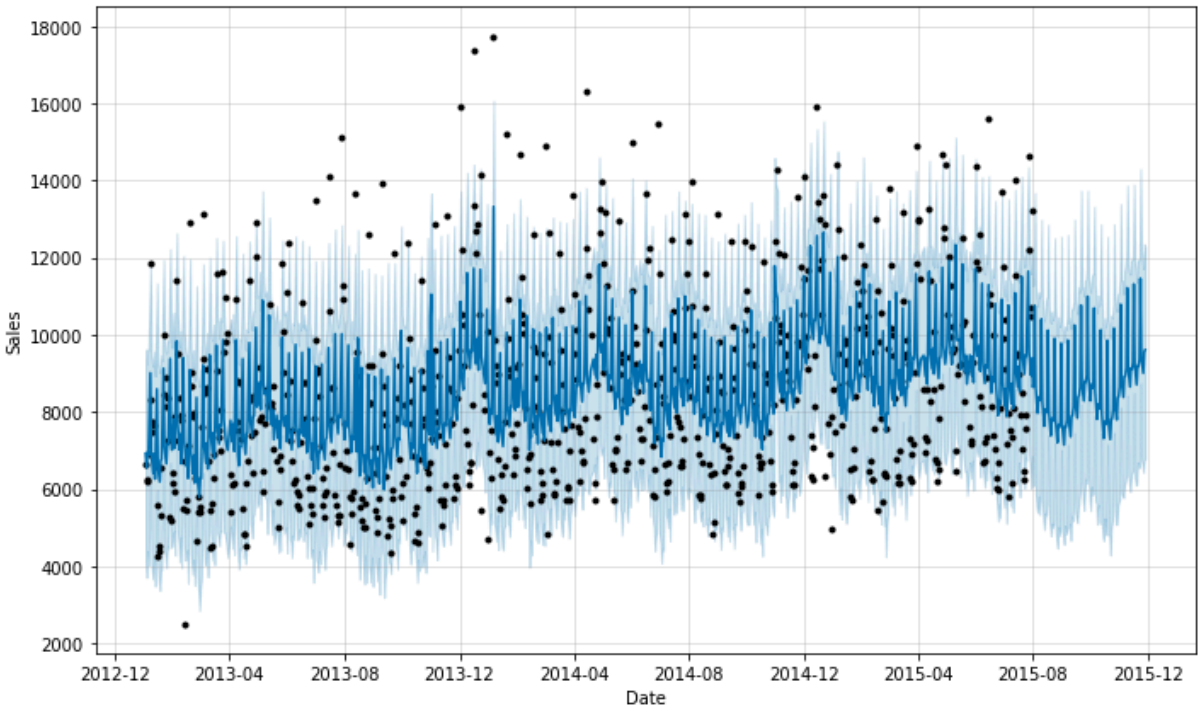
Out[36]:

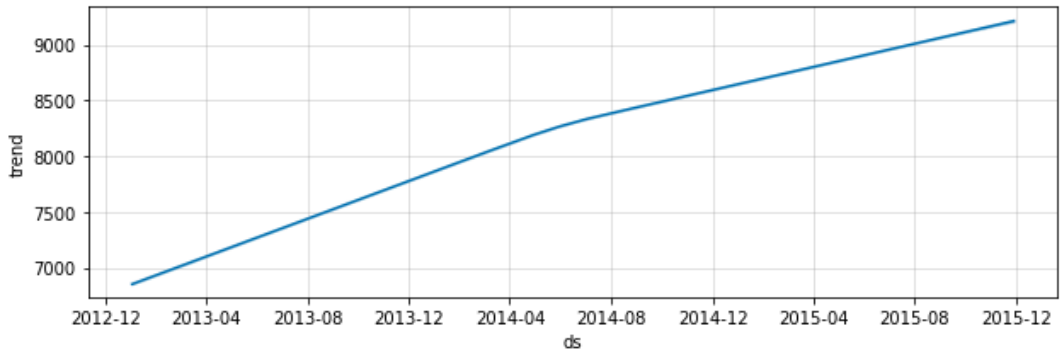
	ds	holiday
0	2014-10-03	state_holiday
1	2013-10-03	state_holiday
2	2015-06-04	state_holiday

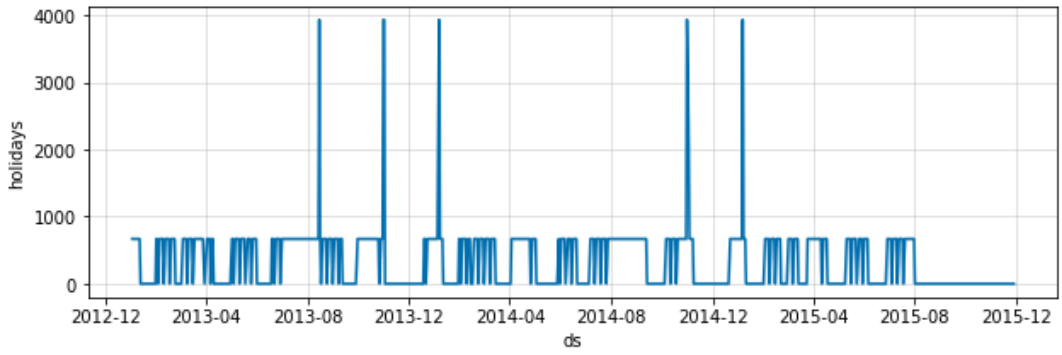
In [37]:
```
# Predictions

# Creating a df with sales data from one specific store and period, considering the holidays:

selected_store = 250
periods = 120  #future days
holidays = holidays_all

sales_store = all_dataset[all_dataset['Store'] == selected_store]

# For the fbprophet, df must have columns "ds" and "y" with the dates and values respectively.

sales_store = sales_store[['Date', 'Sales']].rename(columns = {'Date' : 'ds', 'Sales' : 'y'})
sales_store = sales_store.sort_values('ds')

# Training the model and predicting for this specific store

model = Prophet(holidays=holidays_all)
model.fit(sales_store)
future_df = model.make_future_dataframe(periods = periods)
predictions = model.predict(future_df)
```

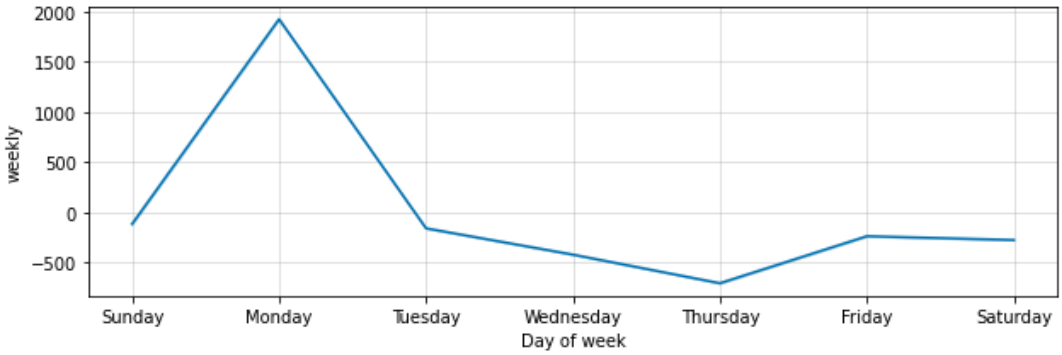
```
Initial log joint probability = -13.7002
    Iter      log prob        ||dx||      ||grad||       alpha      alpha0  # evals  Notes
      99       1266.68   1.19646e-05       76.1964     0.05803           1      121
    Iter      log prob        ||dx||      ||grad||       alpha      alpha0  # evals  Notes
     116       1266.69   9.91241e-05        93.983   1.275e-06       0.001      177  LS failed, Hessian re
set
     170        1266.7   3.79709e-06       74.5836   4.783e-08       0.001      286  LS failed, Hessian re
set
     193       1266.71   6.65249e-08       61.4975       0.441           1      316
Optimization terminated normally:
  Convergence detected: relative gradient magnitude is below tolerance
```

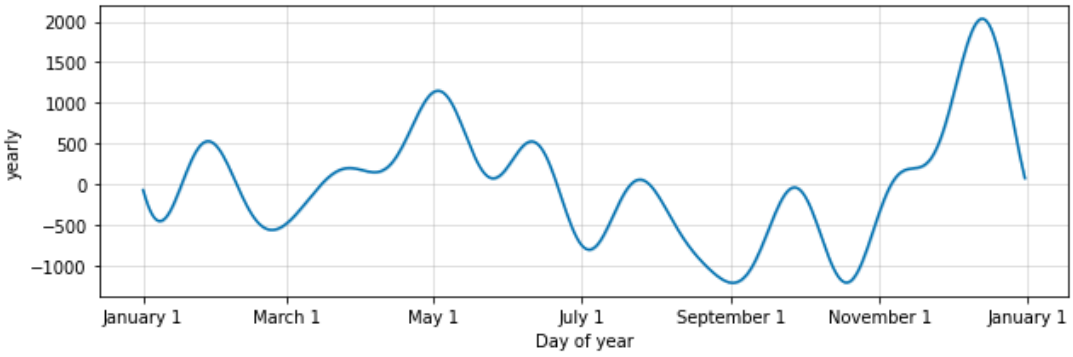
In [38]:
```
# Plotting the result

ax = model.plot(predictions, xlabel = 'Date', ylabel = 'Sales')
ax2 = model.plot_components(predictions)
```











Conclusion

The project successfully utilized the Facebook Prophet library to predict future sales based on historical data. The model effectively incorporated seasonality, holidays, promotions, and competition into the forecast, providing a comprehensive outlook on sales trends. The results demonstrated the capability of Prophet to handle various external factors and yield reliable predictions. This approach facilitates better decision-making in inventory management, resource allocation, and marketing strategies.

Recommendation

1. Enhance Promotions: Increase the frequency and variety of promotional campaigns during identified sales troughs to boost customer engagement and drive sales.

2. Optimize Inventory: Use forecast insights to fine-tune inventory levels, ensuring high-demand items are always in stock, while minimizing overstock of low-demand items.
3. Improve Customer Experience: Invest in customer service and in-store experience enhancements to increase customer satisfaction and repeat business.
4. Expand Data Sources: Integrate additional data sources such as social media trends, local events, and economic indicators to refine sales predictions and identify new growth opportunities.
