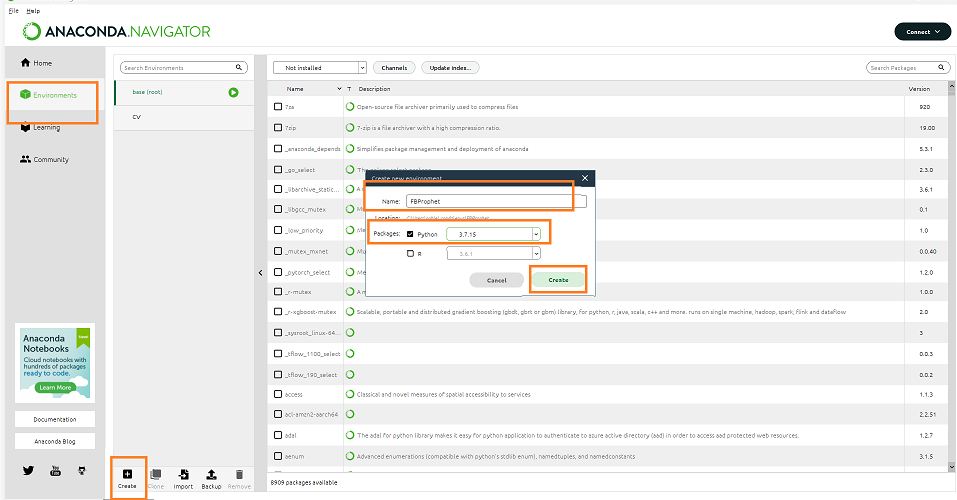
**Sales Analysis**

**Create a New Environment**

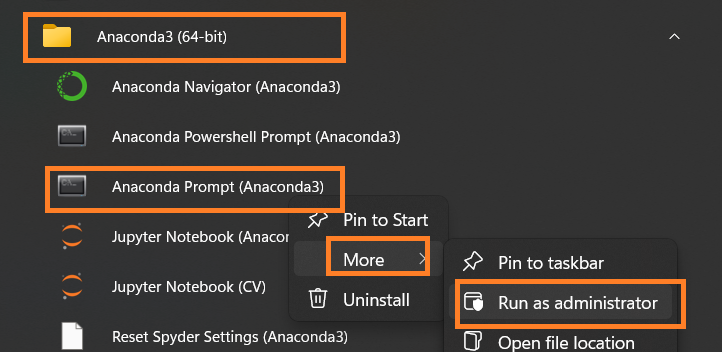
Create a new environment, in anaconda navigator for FBProphet

Use python version 3.7 (Higher versions it won’t work)

****

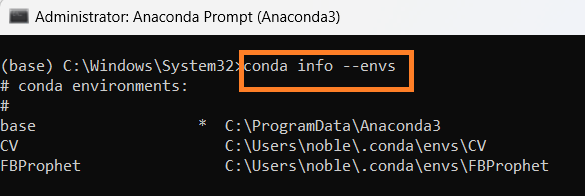
**Open Anaconda Prompt**

Open Anaconda Prompt with Administrator Privilege



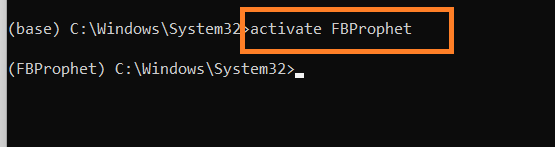
**List all environments**

conda info --envs



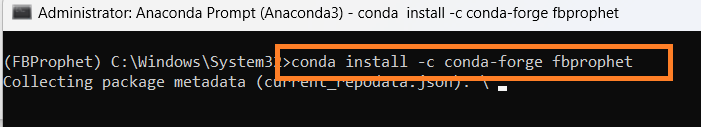
**Activate New Environment**

activate FBProphet

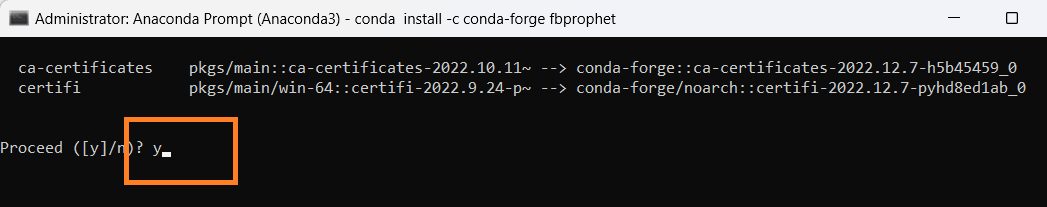


**Install the fbprophet**

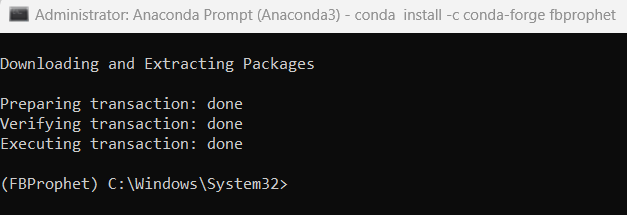
conda install -c conda-forge fbprophet



Proceed y

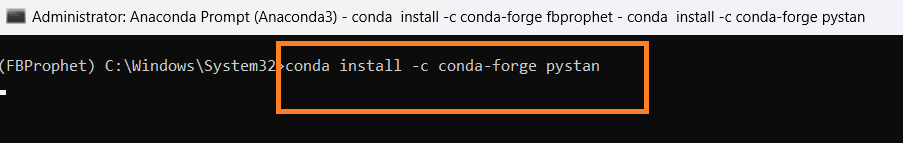


**Installation – Complete**



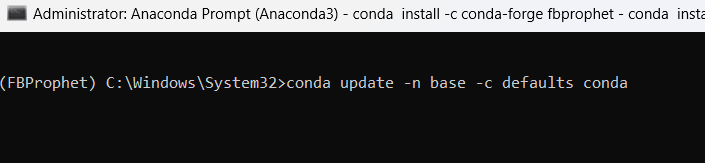
**Install the Following**

conda install -c conda-forge pystan

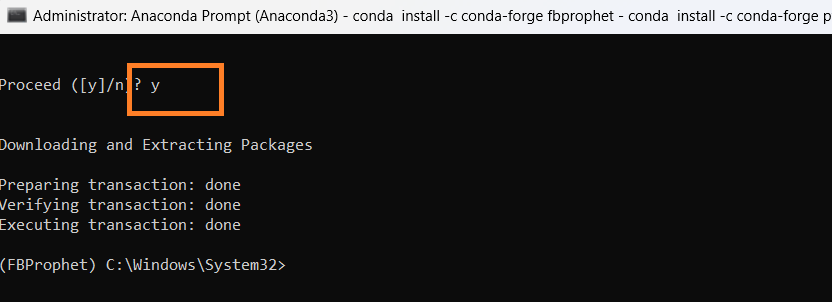


**Install the new version of Conda if any error**

conda update -n base -c defaults conda

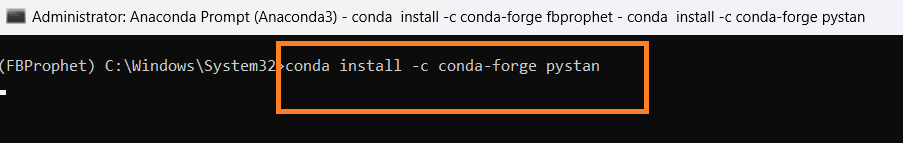


Proceed y

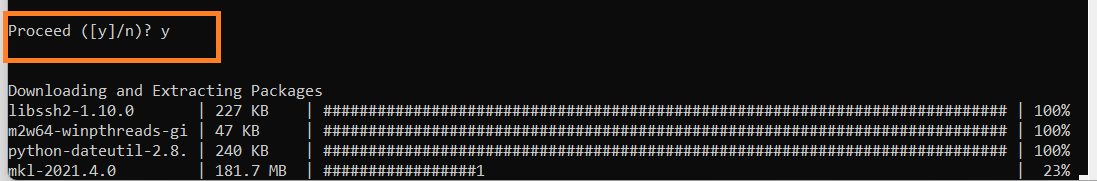


**Install the Following -Pystan Again**

conda install -c conda-forge pystan

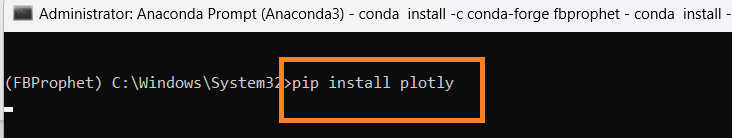


Proceed y /n Enter y



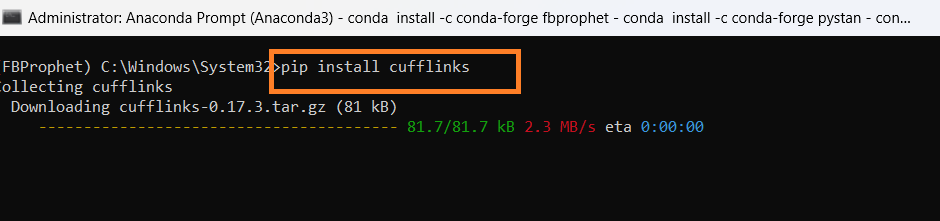
**Install the Following -Plotly**

pip install plotly



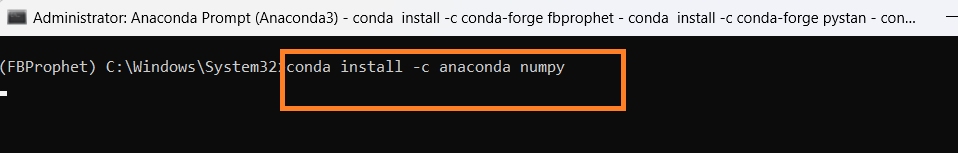
**Install the Following -Cufflinks**

pip install cufflinks



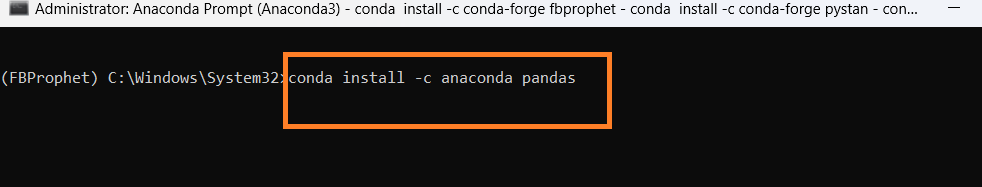
**Install Numpy**

conda install -c anaconda numpy



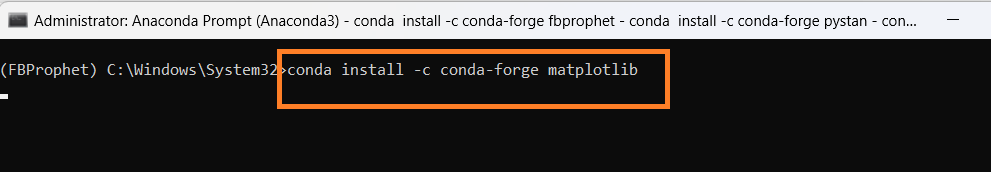
**Install Pandas**

conda install -c anaconda pandas

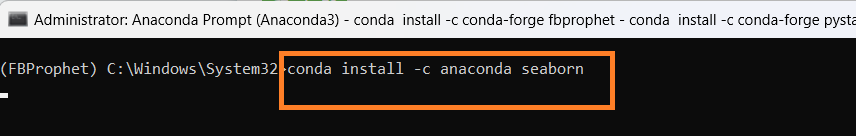


**Install Matplotlib**

conda install -c conda-forge matplotlib



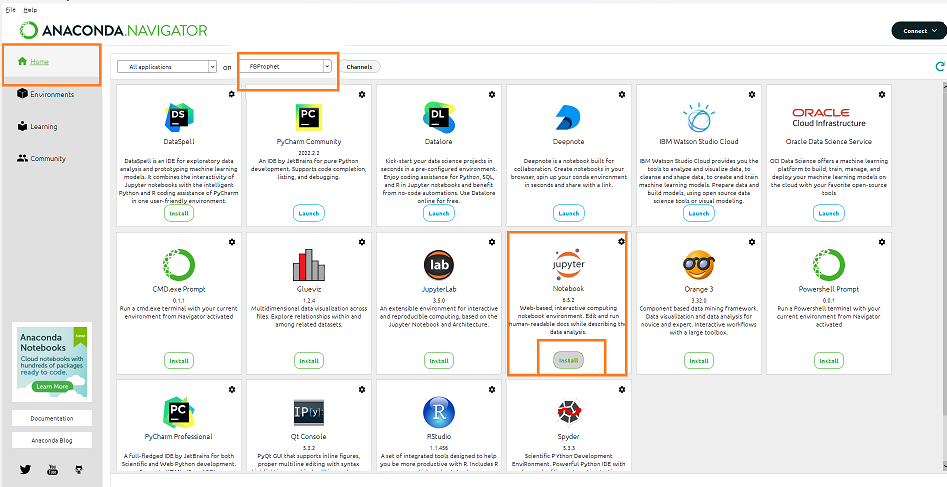
**Install Seaborn**



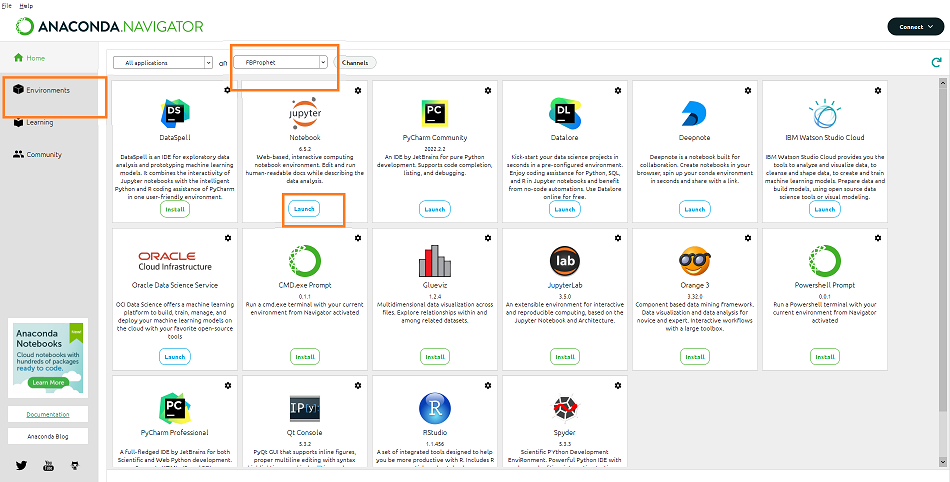
**Open Anaconda Navigator**

Select the new environment created in Anaconda Navigator FBProphet

Click install to install Jupyter Notebook



Then Launch Jupyter Notebook in the new environment



**Import Libraries**

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

from matplotlib.dates import MonthLocator, num2date

from matplotlib.ticker import FuncFormatter

import datetime

import os

%matplotlib inline

from matplotlib.pylab import rcParams

rcParams['figure.figsize'] = 20,10

**Check Current Directory**

print (os.getcwd())

**Change the directory**

os.chdir ('C:\\Noble\\Training\\Top Mentor\\Training\\Presentation\\Project\\Project -6 Sales Department\\')

print (os.getcwd())

**Read Sales Data , display top 5 records**

sales\_train\_df = pd.read\_csv('train.csv')

display (sales\_train\_df.head())

# Almost a million observation

# 1115 unique stores

# Note that sales is the target variable (that's what we are trying to predict)

# 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

**Display the shape (number of rows and columns)**

display (sales\_train\_df.shape)

**Display the bottom 5 records**

display (sales\_train\_df.tail())

**Data Frame Details**

display (sales\_train\_df.info())

# 9 columns in total

# 8 features, each containing 1017209 data points

# 1 target variable (sales)

**Describe Sales Data Frame**

display (sales\_train\_df.describe())

# Average sales amount per day = 5773 Euros, minimum sales per day = 0, maximum sales per day = 41551

# Average number of customers = 633 per day, minimum number of customers = 0, maximum number of customers = 7388

**Import Store Data**

store\_info\_df = pd.read\_csv('store.csv')

display (store\_info\_df.head())

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

# Assortment: describes an assortment level: a = basic, b = extra, c = extended

# CompetitionDistance (meters): distance to closest competitor store

# CompetitionOpenSince [Month/Year]: provides an estimate of the date when competition was open

# Promo2: Promo2 is a continuing and consecutive promotion for some stores - Check is the store participate additional promotions (0 = store is not participating, 1 = store is participating)

# Promo2Since [Year/Week]: date when the store started participating in Promo2

# PromoInterval: Duration of promo in months.Describes the consecutive intervals Promo2 is started, naming the months the 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

**Display the shape (number of rows and columns)**

display (store\_info\_df.shape)

**Store Data Details - Info**

# This dataframe only includes information about the unique 1115 stores that are part of this study

display (store\_info\_df.info())

**Store Data Details -Describe**

display (store\_info\_df.describe())

# on average, the competition distance is 5404 meters away (5.4 kms)

EXPLORE DATASET

**EXPLORE SALES DATA**

**Check number of Null records**

print (sales\_train\_df.isna().sum())

**Plot Missing Data – Heat Map to check missing data, in this case no graph created , no missing data**

# Check if any missing data with seaborn heat hap

# sns.heatmap(sales\_train\_df.isnull(), yticklabels = False, cbar ( Colour bar) = False, cmap="Blues")

sns.heatmap(sales\_train\_df.isnull(), yticklabels = False, cbar = False, cmap="Blues")

plt.show()

**Create Histogram of sales data. Histogram created for numeric columns**

sales\_train\_df.hist(bins = 30, figsize = (20,20), color = 'r')

plt.show()

**# Few observations from Histogram**

#Average 600 customers per day, maximum is 4500

# Data is equally distributed across various Days of the week (~150000 observations x 7 day = ~1.1 million observation)

# Stores are open ~80% of the time

# Data is equally distributed among all stores (no bias)

# Promo #1 was running ~40% of the time

# Average sales around 5000-6000

# School holidays are around ~18% of the time

**Maximum number of customers in a store in a day.**

display (sales\_train\_df['Customers'].max())

**Number of stores open or closed**

closed\_train\_df = sales\_train\_df[sales\_train\_df['Open'] == 0]

open\_train\_df = sales\_train\_df[sales\_train\_df['Open'] == 1]

**Count the number of stores that are open and closed**

print("Total =", len(sales\_train\_df))

print("Number of closed stores =", len(closed\_train\_df))

print("Number of open stores =", len(open\_train\_df))

**Remove closed store sales records**

# Only keep open stores and remove closed stores, since there no sales for closed stores, sales are 0 for closed store

sales\_train\_df = sales\_train\_df[sales\_train\_df['Open'] == 1]

**Display the data after removing closed store data**

display(sales\_train\_df)

**Drop the open column since it has no meaning now**

sales\_train\_df.drop(['Open'], axis=1, inplace=True)

display(sales\_train\_df)

**Describe the data**

display (sales\_train\_df.describe())

# Average sales = 6955 average number of customers = 762 (went up)

**EXPLORE STORES INFORMATION DATA**

**Check missing data**

display (store\_info\_df.isna().sum())

**Heat Map to check missing Data**

# Check missing data in the store information data frame

sns.heatmap(store\_info\_df.isnull(), yticklabels = False, cbar = False, cmap="Blues")

plt.show()

**Display Records with missing column - Column - CompetitionDistance**

# Display records with missing values in the 'CompetitionDistance' column

# Only 3 rows are missing

display (store\_info\_df[store\_info\_df['CompetitionDistance'].isnull()])

**Display Records with missing column - Column - CompetitionOpenSinceMonth**

# Display records with missing values in the 'CompetitionOpenSinceMonth' column

# Many rows are missing = 354 (almost one third of the 1115 stores)

display (store\_info\_df[store\_info\_df['CompetitionOpenSinceMonth'].isnull()])

**Display all records with Promo2 = 0**

display (store\_info\_df[ store\_info\_df['Promo2'] == 0])

# It seems like if 'promo2' is zero, 'promo2SinceWeek', 'Promo2SinceYear', and 'PromoInterval' information is set to zero

**Display Records with missing column - Column - CompetitionOpenSincYear**

# Display records with missing values in the column'CompetitionOpenSinceYear No of records 354'

display (store\_info\_df[store\_info\_df['CompetitionOpenSinceYear'].isnull()])

**Display Records with missing values in column -CompetitionOpenSincMonth**

# Display records with missing values in the column'CompetitionOpenSinceMonth No of records 354'

display (store\_info\_df[store\_info\_df['CompetitionOpenSinceMonth'].isnull()])

**Update Null Columns with 0**

# There are 354 rows where 'CompetitionOpenSinceYear' and 'CompetitionOpenSinceMonth' is missing

# Update the following columns to zeros , if null

str\_cols = ['Promo2SinceWeek', 'Promo2SinceYear', 'PromoInterval', 'CompetitionOpenSinceYear', 'CompetitionOpenSinceMonth']

for str in str\_cols:

store\_info\_df [str].fillna(0, inplace = True)

**Check for NULL values**

display (store\_info\_df.isna().sum())

**Create heat map to check for NULL values**

sns.heatmap(store\_info\_df.isnull(), yticklabels = False, cbar = False, cmap="Blues")

plt.show()

**Missing Values column -** **CompetitionDistance'**

# There are 3 rows with 'competitionDistance' values missing, let's fill them up with with average values of the 'CompetitionDistance' column

store\_info\_df['CompetitionDistance'].fillna(store\_info\_df['CompetitionDistance'].mean(), inplace = True)

**Check for NULL values**

display (store\_info\_df.isna().sum())

**Create heat map to check for NULL values**

sns.heatmap(store\_info\_df.isnull(), yticklabels = False, cbar = False, cmap="Blues")

plt.show()

**Create histogram to View Data**

store\_info\_df.hist(bins = 30, figsize = (20,20), color = 'r')

plt.show()

# half of stores are involved in promo 2

# half of the stores have their competition at a distance of 0-3000m (3 kms away)

**EXPLORE MERGED DATASET**

**Merge Both Data Frames**

# Let's merge both data frames together based on 'store'

sales\_train\_all\_df = pd.merge(sales\_train\_df, store\_info\_df, how = 'inner', on = 'Store')

**Display Merged Data Frames**

# Create .csv file with merged data set

sales\_train\_all\_df.to\_csv('merged\_test.csv', index=False)

**Create .csv file with merged data set**

sales\_train\_all\_df.to\_csv('merged\_test.csv', index=False)

**Create corelation with Sales Columns**

correlations = sales\_train\_all\_df.corr()['Sales'].sort\_values()

print (correlations)

# Customers and promo are positively correlated with the sales

# Promo2 does not seem to be effective at all

**Create heat Map to check co relation**

correlations = sales\_train\_all\_df.corr()

f, ax = plt.subplots(figsize = (20, 20))

sns.heatmap(correlations, annot = True)

plt.show()

# Customers/Prmo2 and sales are strongly correlated

**Extract Year from Date Column**

# Separate the year from the date column and put it into a separate column

sales\_train\_all\_df['Year'] = pd.DatetimeIndex(sales\_train\_all\_df['Date']).year

display(sales\_train\_all\_df)

**Extract Month and Day Column**

# Same way extract month and day column, add as separate columns

sales\_train\_all\_df['Month'] = pd.DatetimeIndex(sales\_train\_all\_df['Date']).month

sales\_train\_all\_df['Day'] = pd.DatetimeIndex(sales\_train\_all\_df['Date']).day

display(sales\_train\_all\_df)

**Average Sales by Month**

# Let's take a look at the average sales per month

# 'groupby' works great by grouping all the data that share the same month column, then obtain the mean of the sales column

# It looks like sales peak around christmas timeframe

axis = sales\_train\_all\_df.groupby('Month')[['Sales']].mean().plot(figsize = (20,10), marker = 'o', color = 'r')

axis.set\_title('Average Sales Per Month')

plt.show()

**Average Customer by Month**

# It looks like number of customers peak around christmas timeframe

plt.figure()

axis = sales\_train\_all\_df.groupby('Month')[['Customers']].mean().plot(figsize = (20,10), marker = '^', color = 'b')

axis.set\_title('Average Customers Per Month')

plt.show()

**Display the sales and customers per day of the month instead**

# Minimum number of customers are generally around the 24th of the month

# Most customers and sales are around 30th and 1st of the month

ax = sales\_train\_all\_df.groupby('Day')[['Sales']].mean().plot(figsize = (20,5), marker = 'o', color = 'r')

axis.set\_title('Average Sales Per Day')

plt.figure()

ax = sales\_train\_all\_df.groupby('Day')[['Customers']].mean().plot(figsize = (20,5), marker = '^', color = 'b')

axis.set\_title('Average Sales Per Day')

plt.show()

**Display the sales and customers based on day of the week**

# Display the sales and customers for the day of the week (note that 7 = Sunday)

axis = sales\_train\_all\_df.groupby('DayOfWeek')[['Sales']].mean().plot(figsize = (20,10), marker = 'o', color = 'r')

axis.set\_title('Average Sales Per Day of the Week')

plt.figure()

axis = sales\_train\_all\_df.groupby('DayOfWeek')[['Customers']].mean().plot(figsize = (20,10), marker = '^', color = 'b')

axis.set\_title('Average Customers Per Day of the Week')

plt.show()

**Average Sales by Store Type by Day (Store Type : a,b,c,d)**

fig, ax = plt.subplots(figsize=(20,10))

sales\_train\_all\_df.groupby(['Date','StoreType']).mean()['Sales'].unstack().plot(ax=ax)

plt.show()

# Store b has average highest sales and store a has average lowest sales

**Average Sales by Store Type by Day without unstack**

#Plot the above graph without unstack

fig, ax = plt.subplots(figsize=(20,10))

sales\_train\_all\_df.groupby(['Date','StoreType']).mean()['Sales'].plot(ax=ax)

plt.show()

**Graph – Impact of Promo on Sales and Customers – Bar Graph**

plt.figure(figsize=[20,15])

plt.subplot(211)

sns.barplot(x = 'Promo', y = 'Sales', data = sales\_train\_all\_df)

plt.subplot(212)

sns.barplot(x = 'Promo', y = 'Customers', data = sales\_train\_all\_df)

**Graph – Impact of Promo on Sales and Customers – violinplot**

plt.figure(figsize=[20,15])

plt.subplot(211)

sns.barplot(x = 'Promo', y = 'Sales', data = sales\_train\_all\_df)

plt.subplot(212)

sns.barplot(x = 'Promo', y = 'Customers', data = sales\_train\_all\_df)

plt.show()

**Graph – Impact of Promo on Sales and Customers – Boxplot**

plt.figure(figsize=[20,15])

plt.subplot(211)

sns.violinplot(x = 'Promo', y = 'Sales', data = sales\_train\_all\_df)

plt.subplot(212)

sns.violinplot(x = 'Promo', y = 'Customers', data = sales\_train\_all\_df)

plt.show()

# TRAIN THE MODEL

**Import Library**

from fbprophet import Prophet

**Create the Function for Forecasting, prediction based on the specified store ie store id**

def sales\_prediction(Store\_ID, sales\_df, periods):

# Function that takes in the data frame, storeID, and number of future period forecast

# The function then generates date/sales columns in Prophet format ie Rename Date column to ds and Sales to y

# The function then makes time series predictions

sales\_df = sales\_df[ sales\_df['Store'] == Store\_ID ]

# Create the data frame by selecting the data based on selected store id

sales\_df = sales\_df[['Date', 'Sales']].rename(columns = {'Date': 'ds', 'Sales':'y'})

# Rename the column

sales\_df = sales\_df.sort\_values('ds')

model = Prophet() # Create the model

model.fit(sales\_df)

future = model.make\_future\_dataframe(periods=periods)

forecast = model.predict(future)

figure = model.plot(forecast, xlabel='Date', ylabel='Sales')

figure2 = model.plot\_components(forecast)

**Call the function to predict for next 60 periods**

sales\_prediction(10, sales\_train\_all\_df, 60)

**Predict the sales for next 60 days for store 6 on State Holiday and School Holiday**

* State Holiday: indicates a state holiday. Normally all stores, with few exceptions, are closed on state holidays. Note that all schools are closed on public holidays and weekends. a = public holiday, b = Easter holiday, c = Christmas, 0 = None
* School Holiday: indicates if the (Store, Date) was affected by the closure of public schools

**Updated Function to predict on Holidays- Included one more input holidays**

def sales\_prediction(Store\_ID, sales\_df, holidays, periods):

# Function that takes in the storeID and returns two date/sales columns in Prophet format

# Format data to fit prophet

sales\_df = sales\_df[ sales\_df['Store'] == Store\_ID ]

sales\_df = sales\_df[['Date', 'Sales']].rename(columns = {'Date': 'ds', 'Sales':'y'})

sales\_df = sales\_df.sort\_values('ds')

model = Prophet(holidays = holidays)

model.fit(sales\_df)

future = model.make\_future\_dataframe(periods = periods)

forecast = model.predict(future)

figure = model.plot(forecast, xlabel='Date', ylabel='Sales')

figure2 = model.plot\_components(forecast)

**Get all the dates related to school holidays**

school\_holidays = sales\_train\_all\_df[sales\_train\_all\_df['SchoolHoliday'] == 1].loc[:, 'Date'].values

display (school\_holidays.shape)

**Print school holidays**

display(school\_holidays)

**Get all the dates related to state holidays**

state\_holidays = sales\_train\_all\_df [ (sales\_train\_all\_df['StateHoliday'] == 'a') | (sales\_train\_all\_df['StateHoliday'] == 'b') | (sales\_train\_all\_df['StateHoliday'] == 'c') ].loc[:, 'Date'].values

display (state\_holidays.shape)

**Print State Holidays**

state\_holidays = pd.DataFrame({'ds': pd.to\_datetime(state\_holidays),

'holiday': 'state\_holiday'})

display(state\_holidays )

**Print School Holidays**

school\_holidays = pd.DataFrame({'ds': pd.to\_datetime(school\_holidays),

'holiday': 'school\_holiday'})

display (school\_holidays)

**Display School and State Holidays**

# Concat state and scholl holidays

school\_state\_holidays = pd.concat((state\_holidays, school\_holidays))

display (school\_state\_holidays)

**Sales Prediction for 60 Day for School and state holidays**

sales\_prediction(6, sales\_train\_all\_df, school\_state\_holidays, 60)