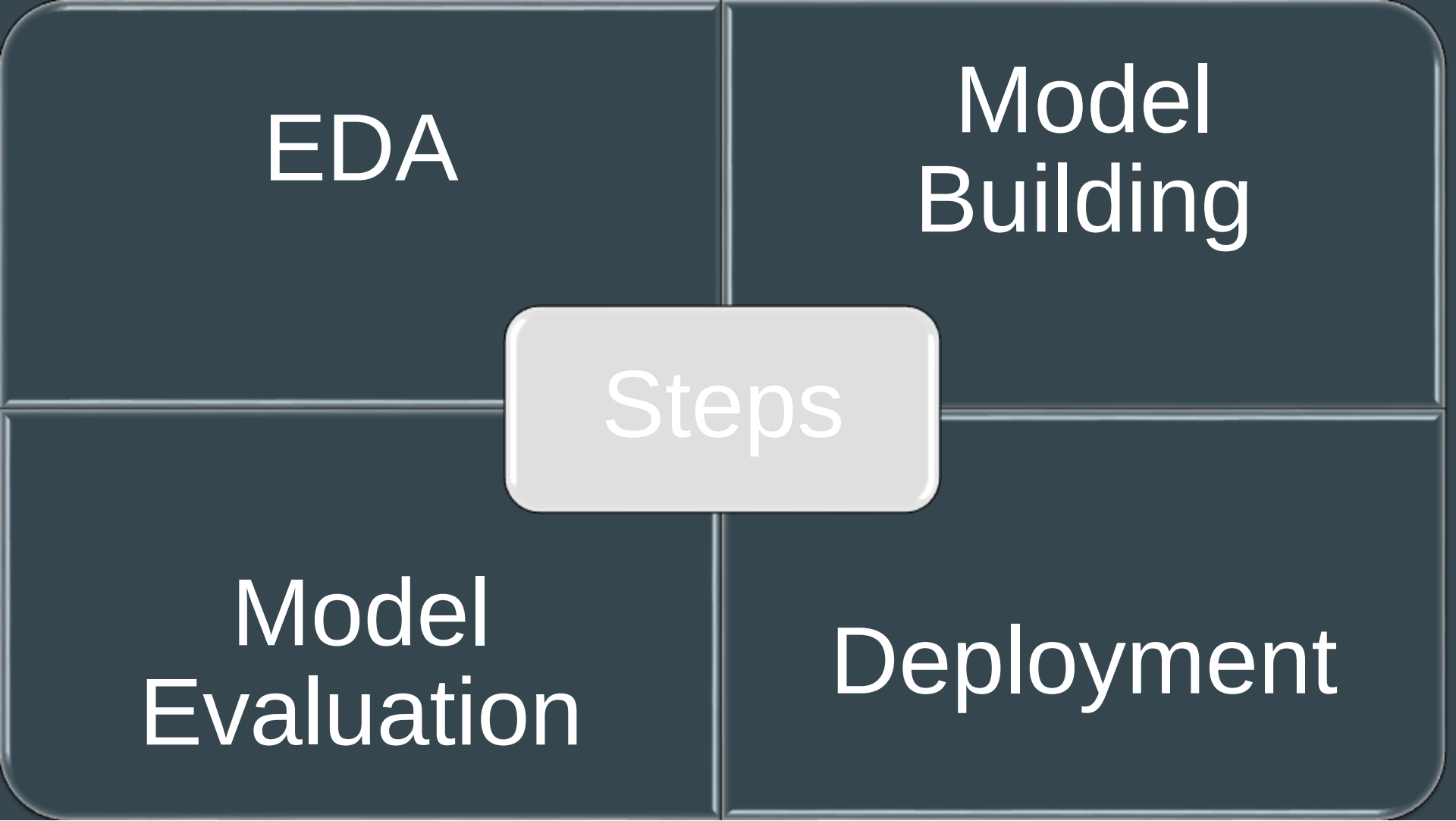
PROJECT:-
DEMAND AND FORECASETING

Presented By:-
Anand Kumar

Objective:-

- This Demand Forecasting will help businesses to predict the quantity of goods and services, which will be demanded by consumers. If we forecast the customer demand for the future, it will be useful to make good business and that is where Data Scientist’s or any person who will handle the data will come into picture.

- We had a requirement related to forecasting the demand from a business point, and found sample data from the web. We will be having a sample of 5 years of Historical data and asking to forecast the next year.


EDA
Model Building
Steps
Model Evaluation
Deployment

Exploratory Data Analysis

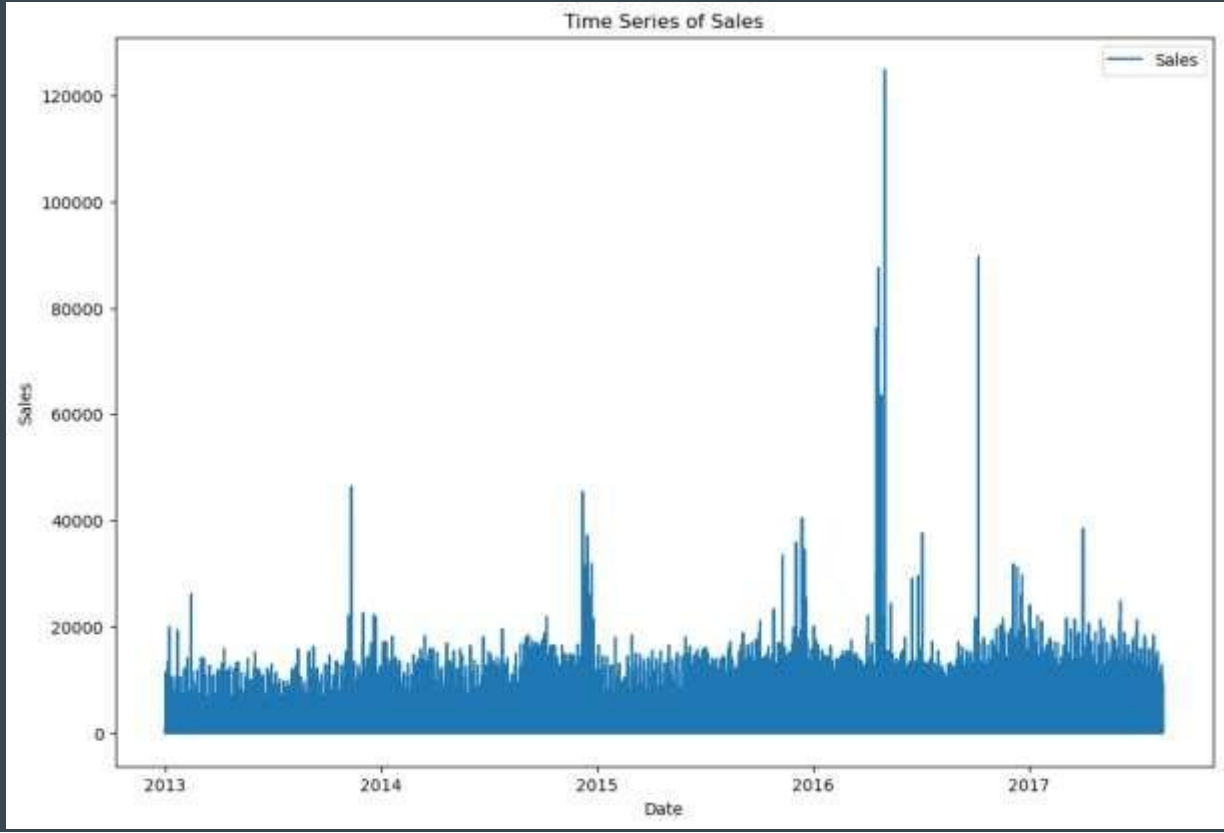
Dataset Overview:

- The training dataset provided for this demand forecasting project contains a time series of features including 'store_nbr', 'family', 'onpromotion', and the target variable 'sales'. The dataset is structured as follows:
- **Shape of Dataset:** The dataset comprises a total of 3,000,888 rows and 6 columns.
- **Unique Store Number:** There are 54 unique store numbers represented in the dataset.
- **Unique Store Families:** The dataset covers 33 unique store families.
- **Missing Values:** There are no missing values present in the dataset.
- **Column Descriptions:**

1. **id:** A unique identifier for each row in the dataset.
2. **date:** The date on which the sales data is recorded. This column establishes the time series aspect of the dataset.
3. **store_nbr:** An identifier for the store where the products are sold.
4. **family:** The category or family to which the product belongs. It represents the type of product being sold.
5. **sales:** The target variable. It represents the total sales of a particular product family at a specific store on a given date. The values may include fractional quantities since products can be sold in fractional units.
6. **onpromotion:** A binary indicator (0 or 1) that represents whether items in a product family were being promoted at a store on a particular date.

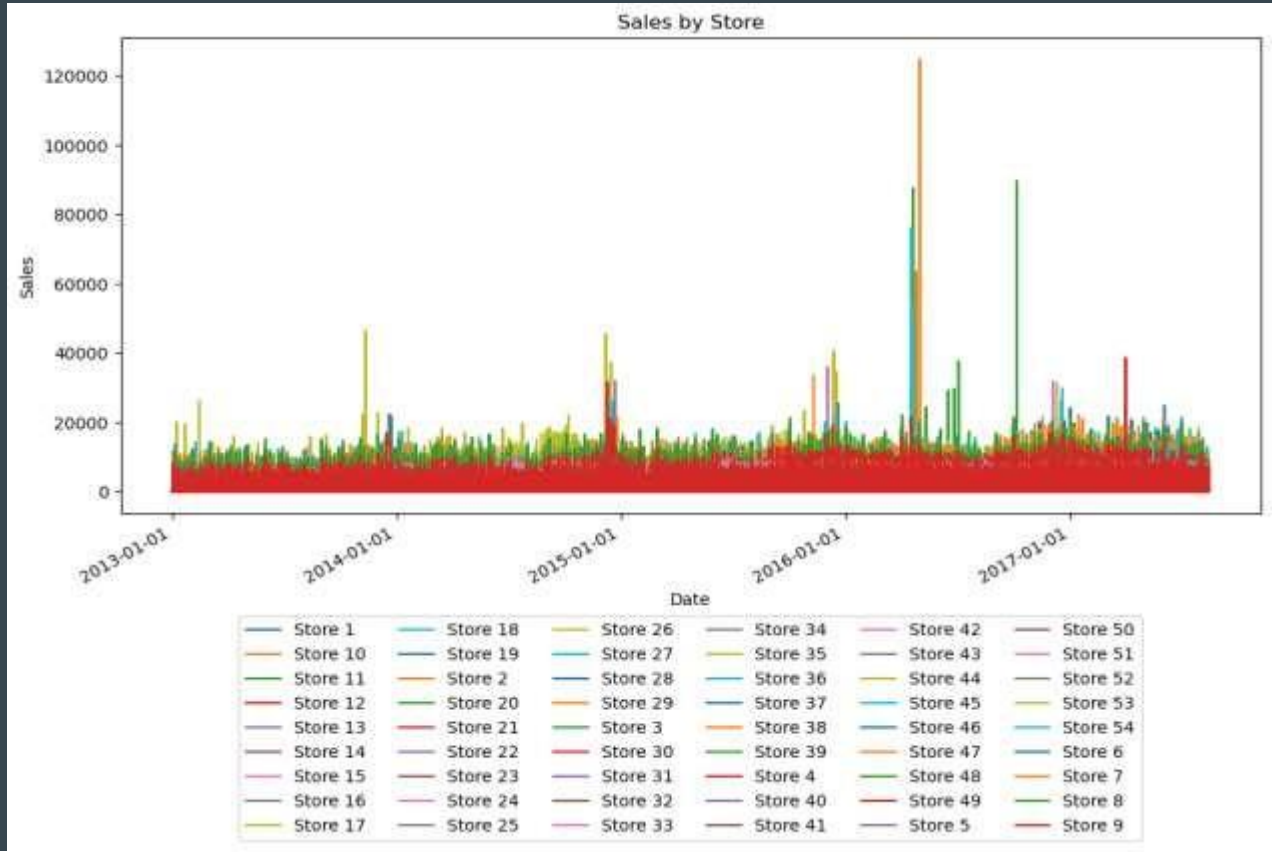
TIME SERIES ANALYSIS

- Time series graph is used to find out trend and seasonality of the data.
- In this time series Analysis X-axis is Date and Y-axis is Sales
- In this graph we analyze that the higher sales we got between the year 2016 and 2017.



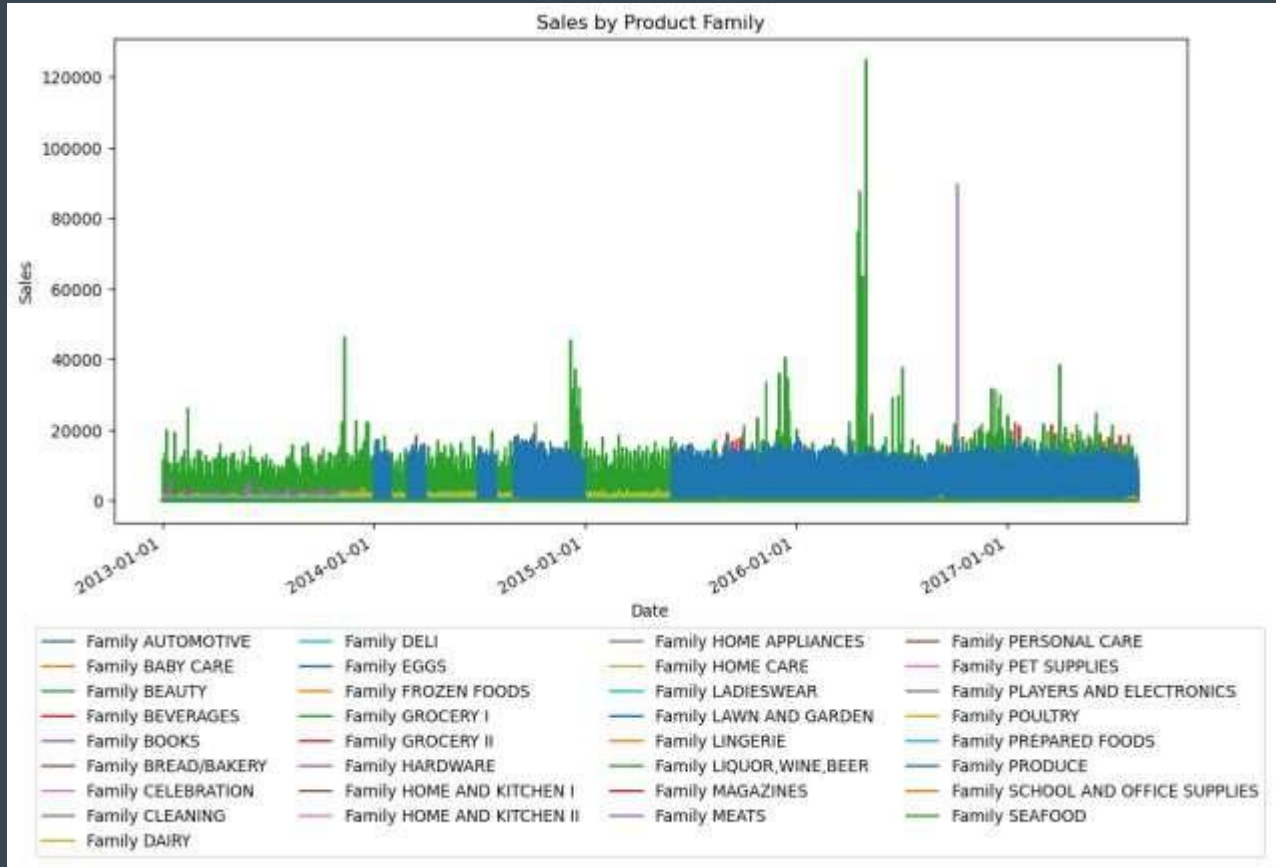
Plot Sales by Store and By Product Family

- This is the graph of year wise sales of the store .
- It represents in one year how much sales by the particular store.
- Every store is represented by the different color to uniquely identify the store



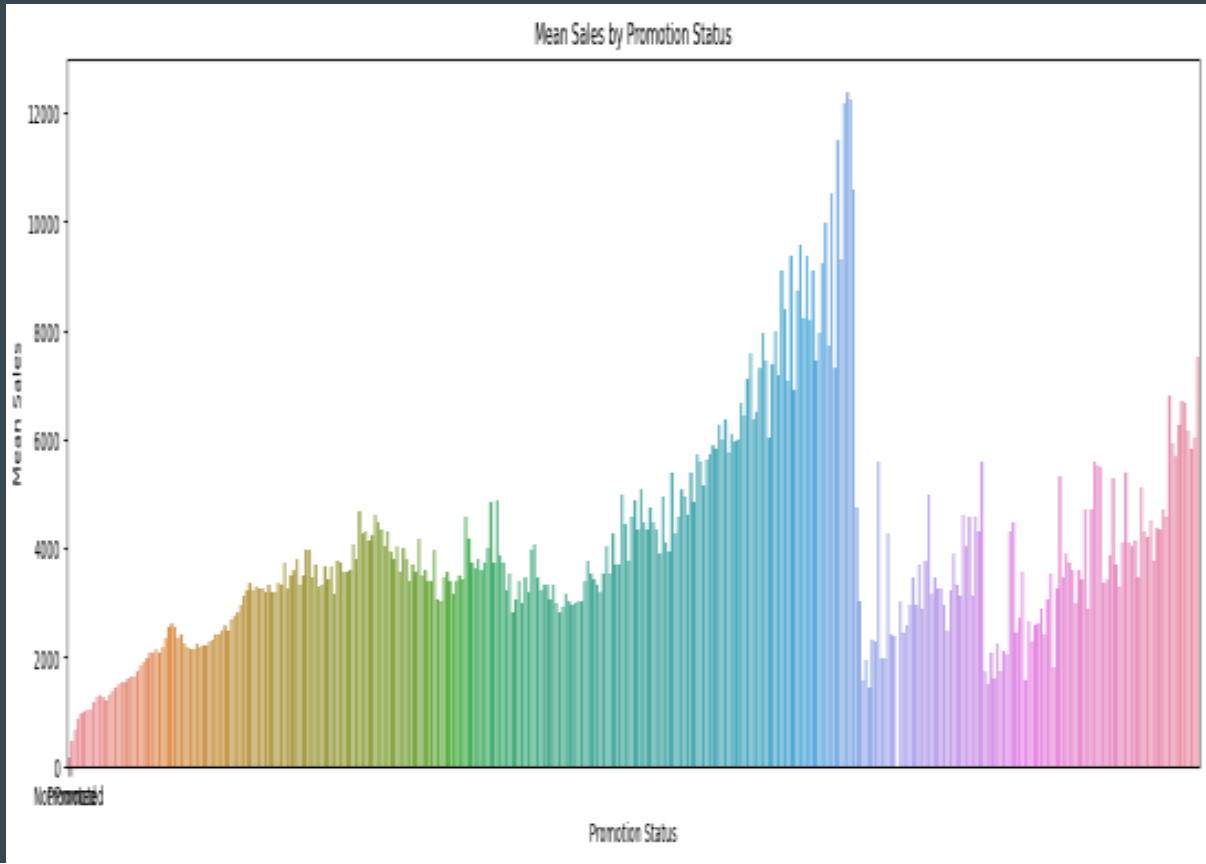
Plot Sales By Product Family

- This graph represents the sales by the product family throughout the year.
- For every product family represented by different color graph
- In this graph x-axis is represented by Date and In y-axis represented by Sales of each product family



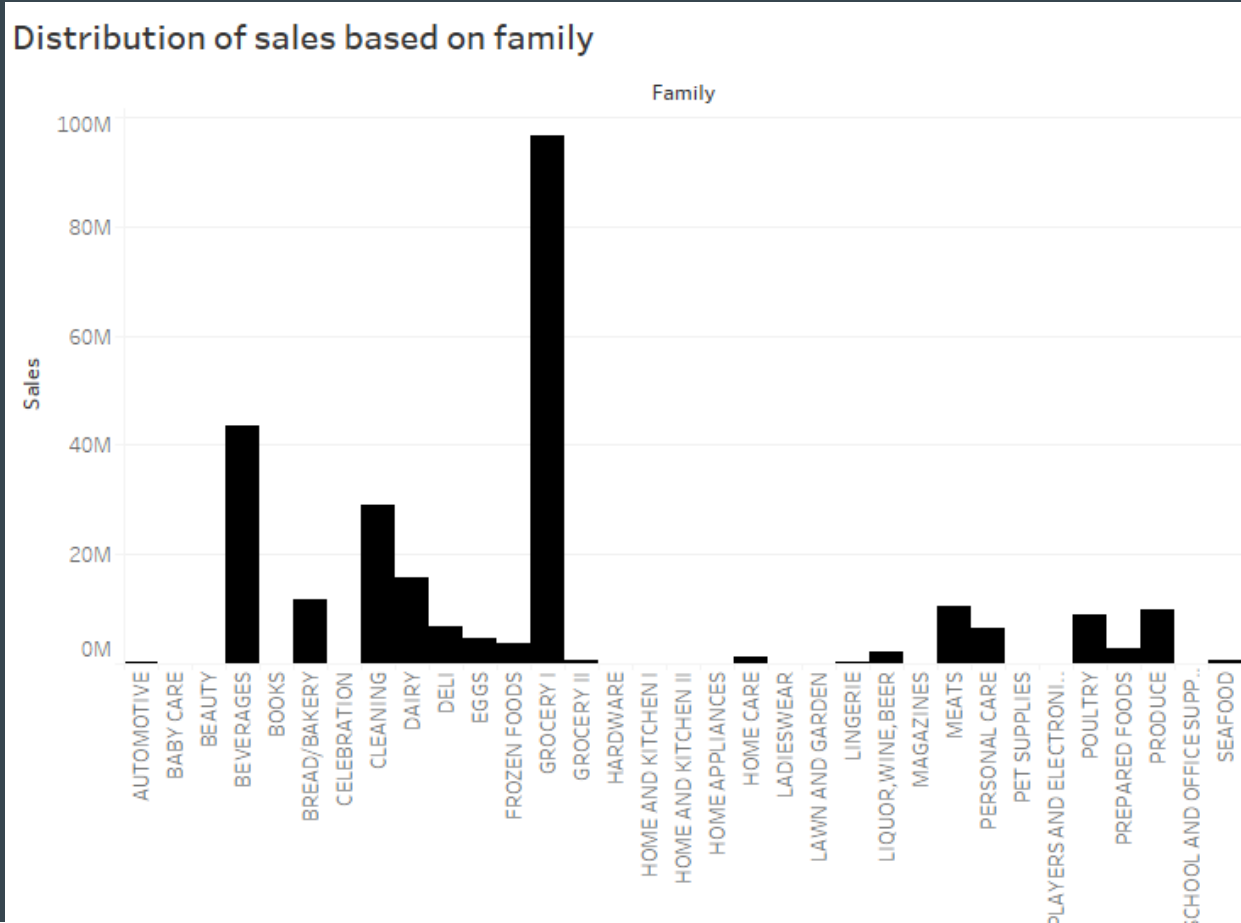
Mean Sales by Promotion Sales

- This graph represents the Promotion by sales
- For It is showing that before promotion how was the sales of product and after promotion applied on product the how sales increased
- Yeaar 2013 no promotion so sales was low but when promotion introduced in year 2014 then sales increased



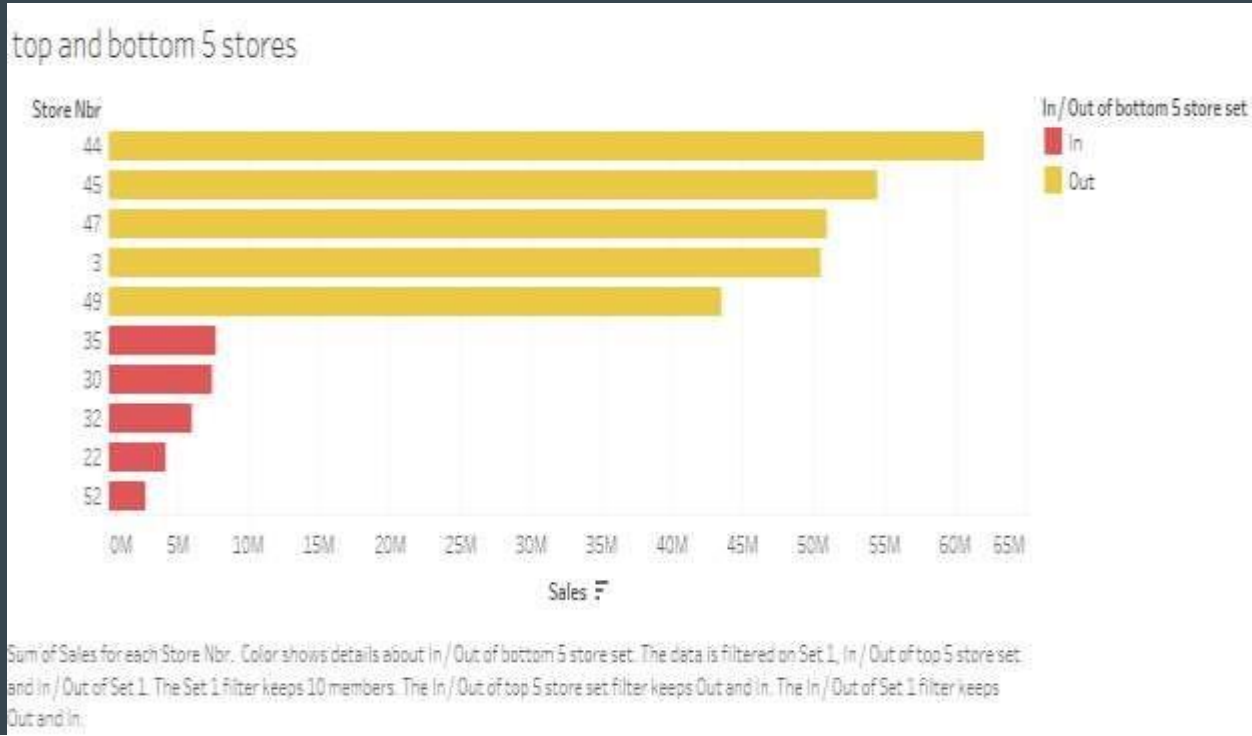
Distribution of sales based on family

- This is the graph represents how much sales done by the particular product.
- In this graph the most sold product is grocery .
- And there is no sales of the home and kitchen and home appliances.
- So we can conclude from this graph is we have to focus on the least salable products rather than the highest saleable product so we can increase the sale of the that product also.



Distribution of sales based on family

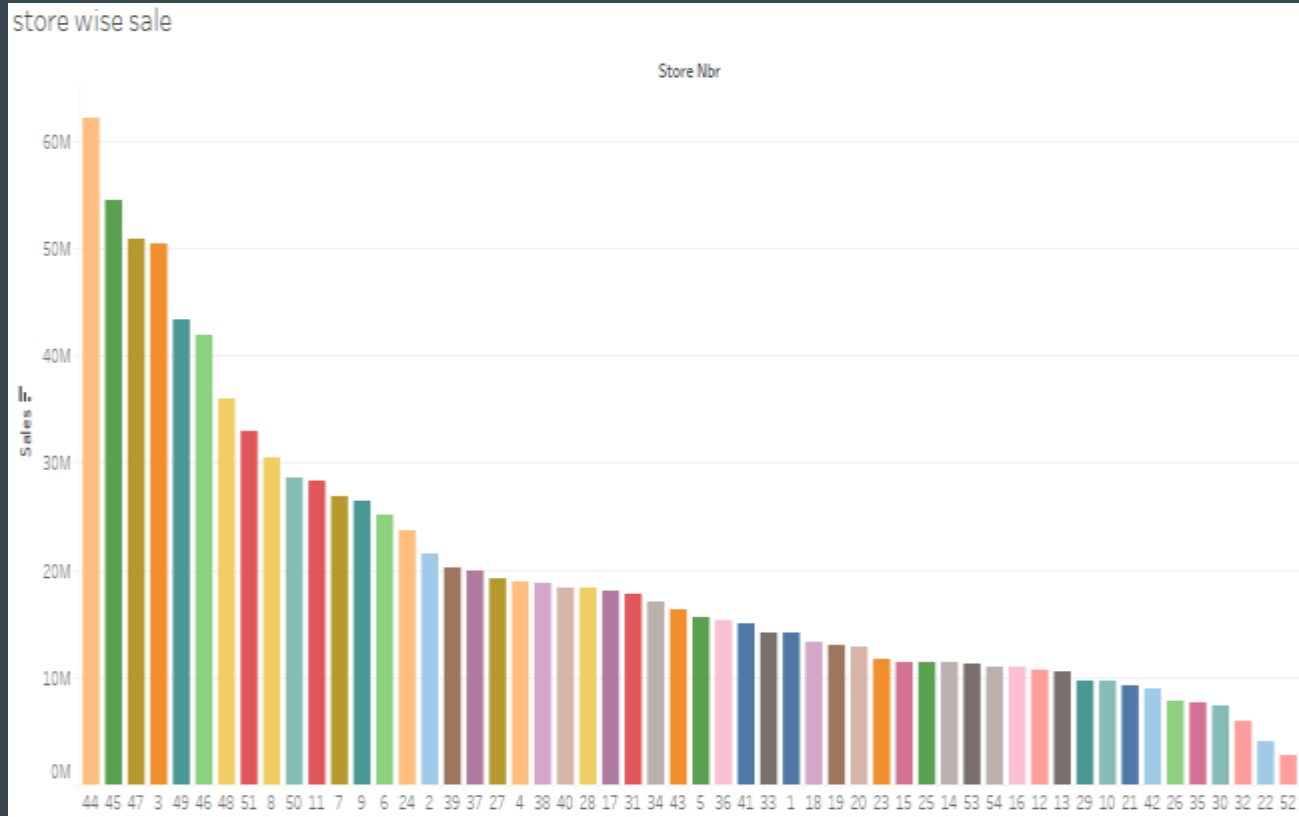
- This graph represents top 5 stores which done the highest sale from all the years.
- And it also represents bottom 5 stores which done the least sale from all the years.
- According to this viz Top Five store is 44,45,47,3,49
- Similarly bottom five store where sales is very less is 35,30,32,22,52



Store Wise Sales

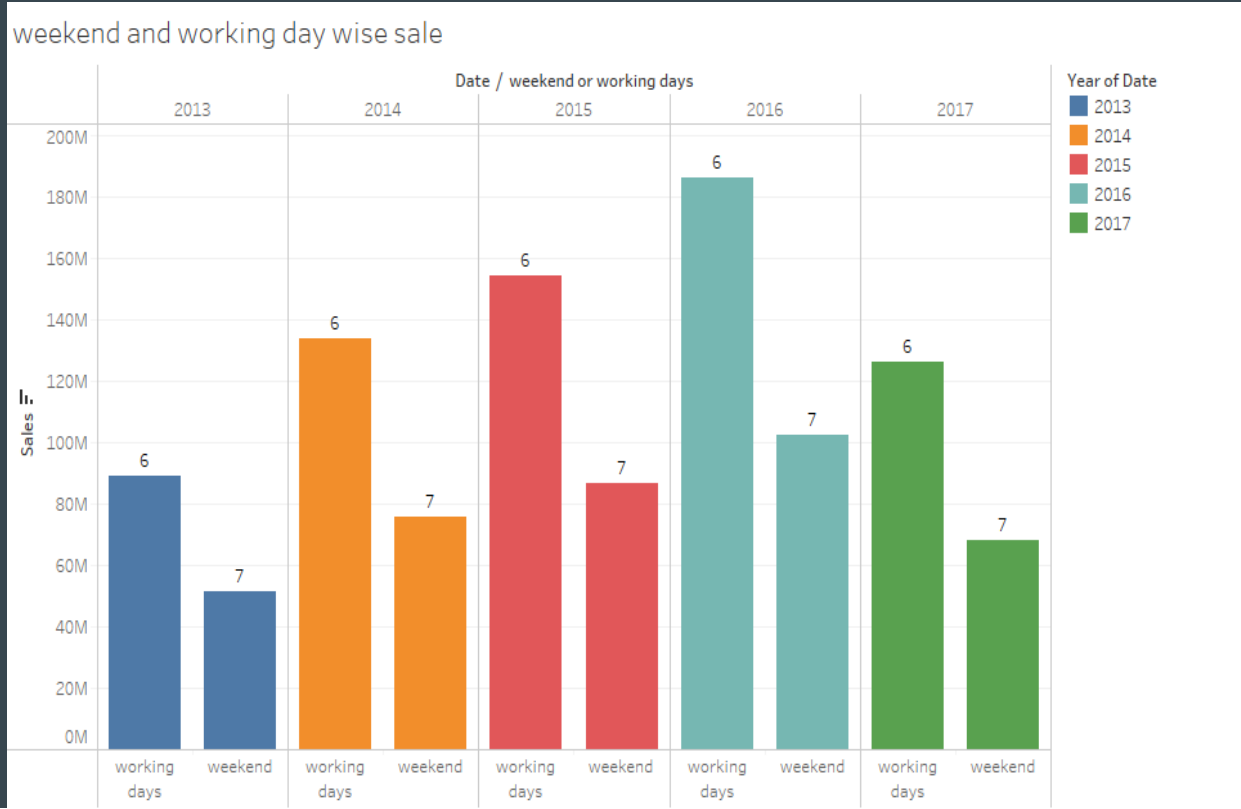
- This graph represents the store wise sales that means it represents how much sales done by the particular product.
- From this graph we can conclude that store 44 done the maximum sale and the store 52 done the minimum sale.
- Every store is represented by the different color.

  Top seller is store number 44,45,47,3



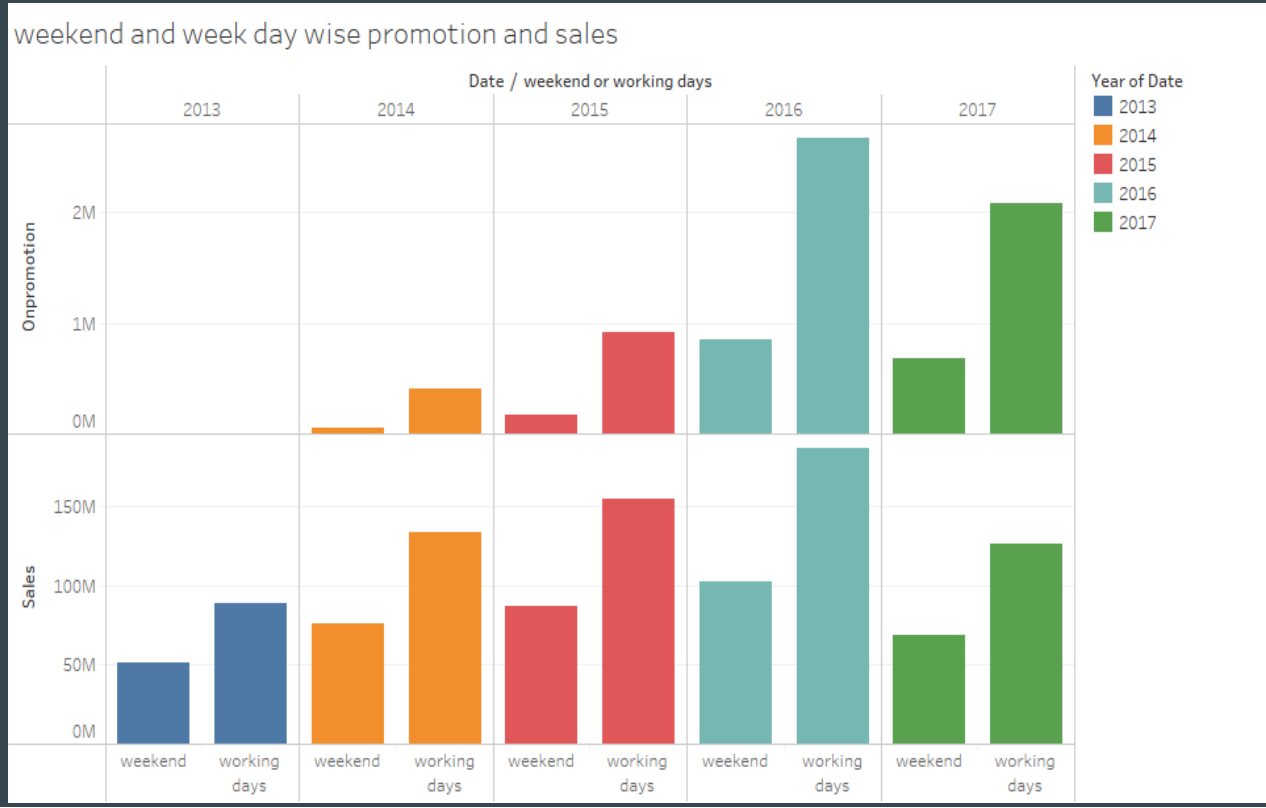
Distribution of sales based on family

- This graph represents the weekday sales and weekend sales.
- Weekend sales is comparatively more than the. weekday sales.
- Saturday and Sunday people are having the week off so they prefer to do shopping in these two days.
- So from this idea we have to promoted the products more on the weekdays to maximise the profit.



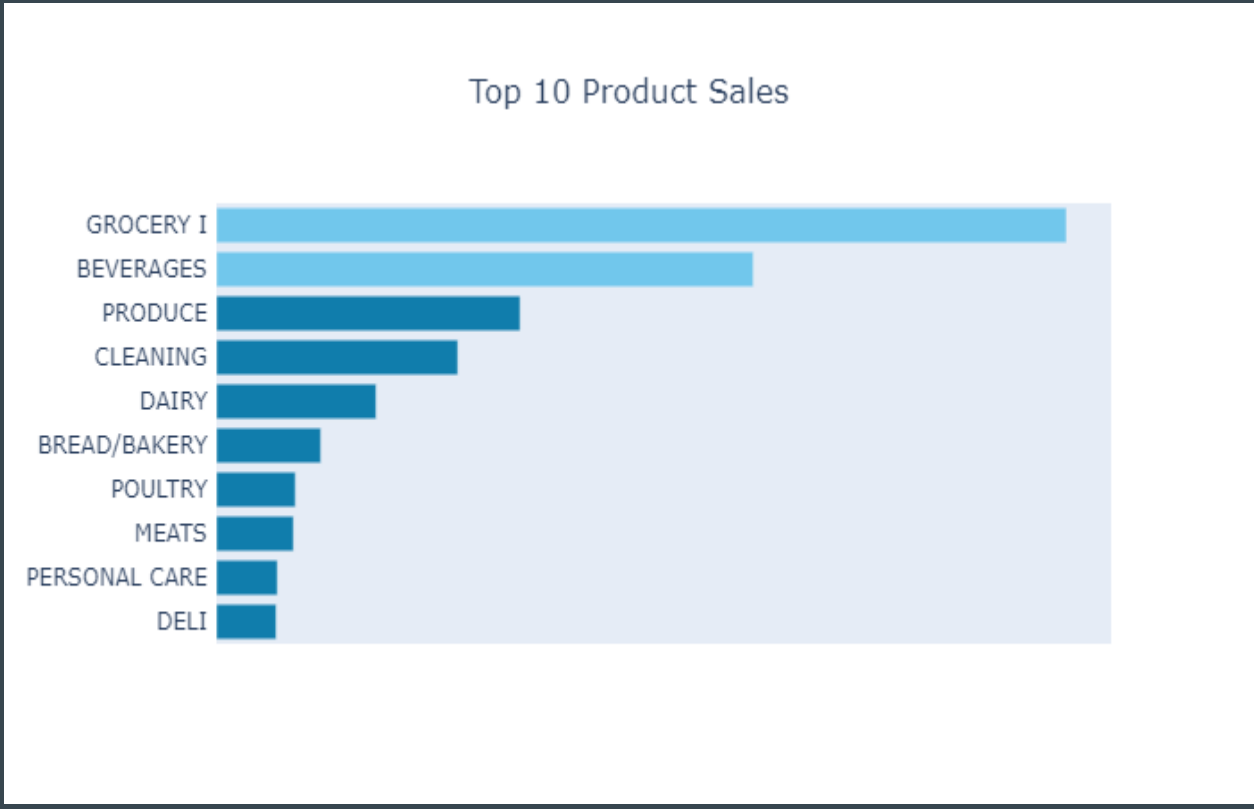
Weekend and Weekday wise promotion and sales

- This graph represents the weekend and weekday wise sales and promotion.
- In 2013 there is no promotion done so the overall sales of the this year is also less.
- Also we can see in this graph there is less promotion done in weekends and most of the promotion done on the product is on the weekdays.



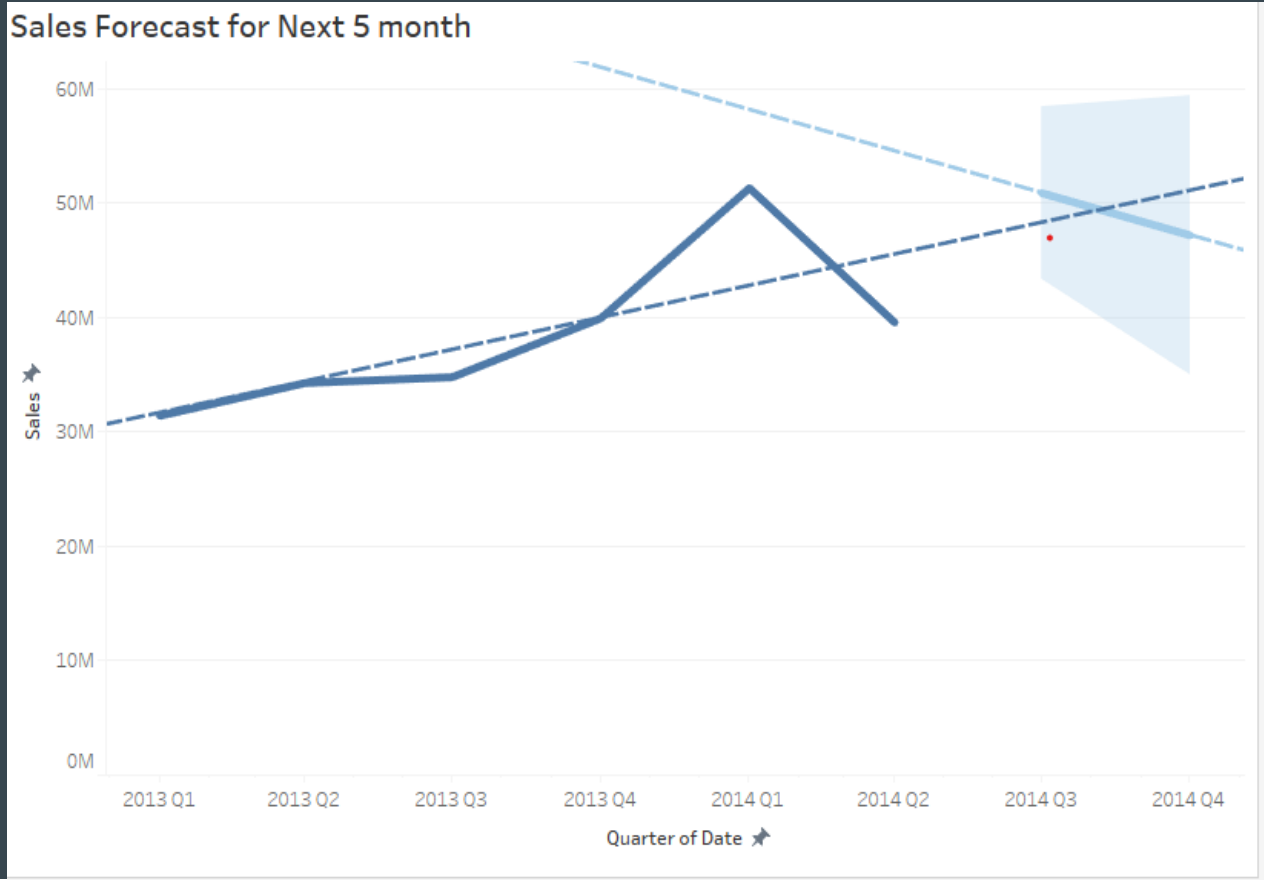
Distribution of sales based on family

- Top Then product that performing will in market well is: -
- Grocery 1,Beverages, Produce and so on.
- These are top then product that sales is high in market as compare to other product



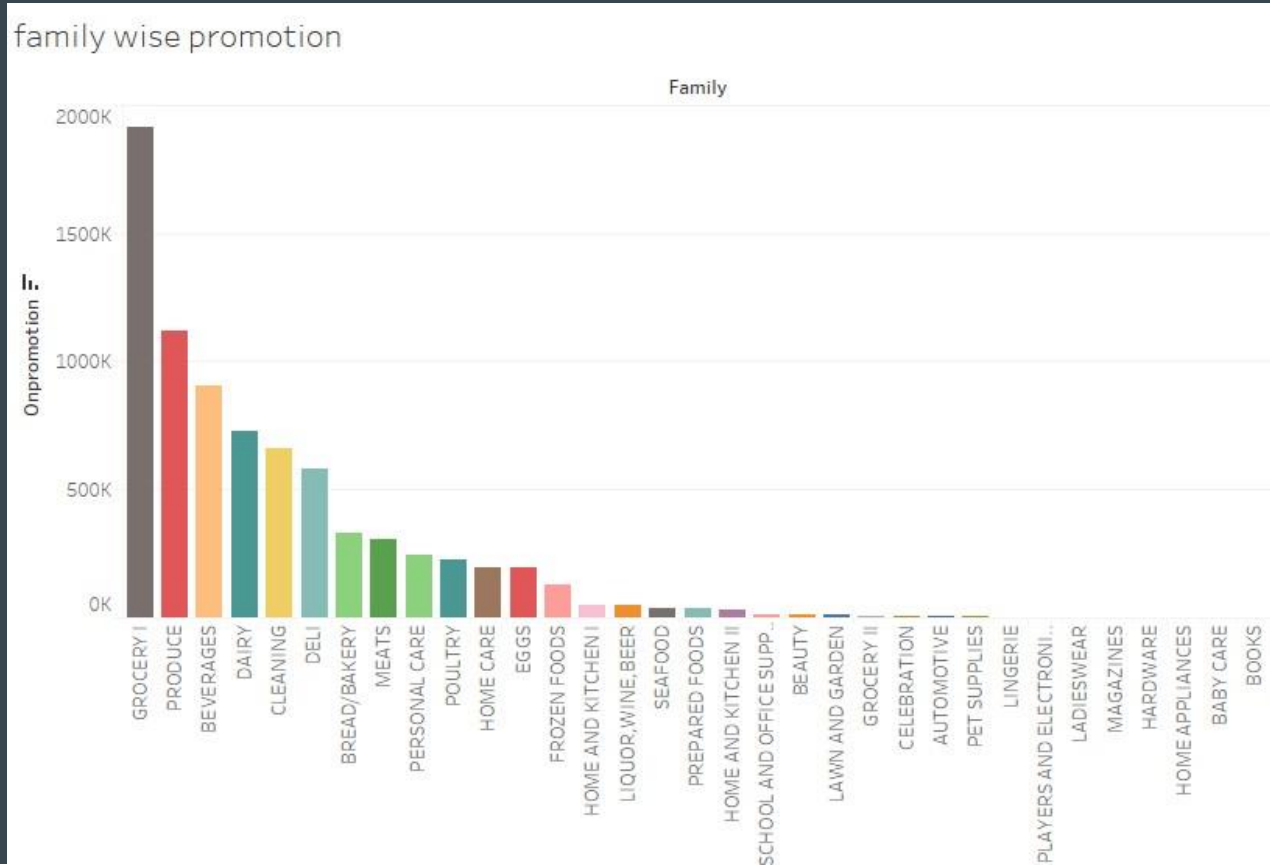
Next 5 Month Forecasting

- This graph learns the overall pattern of the data and predict the next five months sales according to the historical data.
- This forecasting I did through the tableau so it is showing then next five month the sales will be between 40M -50M as per present and past sales data



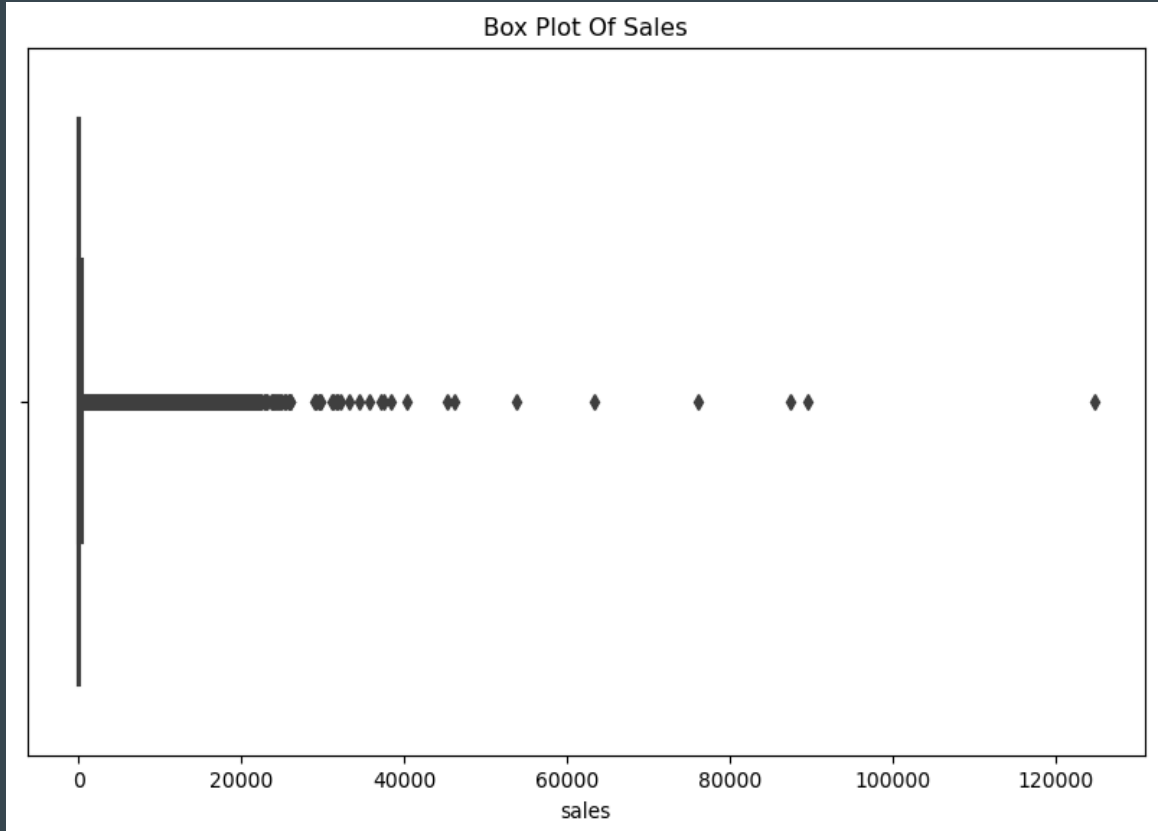
Family Wise Promotion

- This graph represents the family wise promotion.
- That means this graph represents the how much promotion done on the particular product
- On grocery maximum promotion done and we also seen in previous graph most saleable product is also grocery .So we can conclude from this information that more we do the promotion then more we do the sale also



Box plot of sales based on Sales

- This is Box plot using for identification of outlers and according to this graph only one product in our data set that sales is more then 1,20,000.
- There are multiple outliers that showing greater then approx. 25000
- Maximum sales is under approx. is below 25000

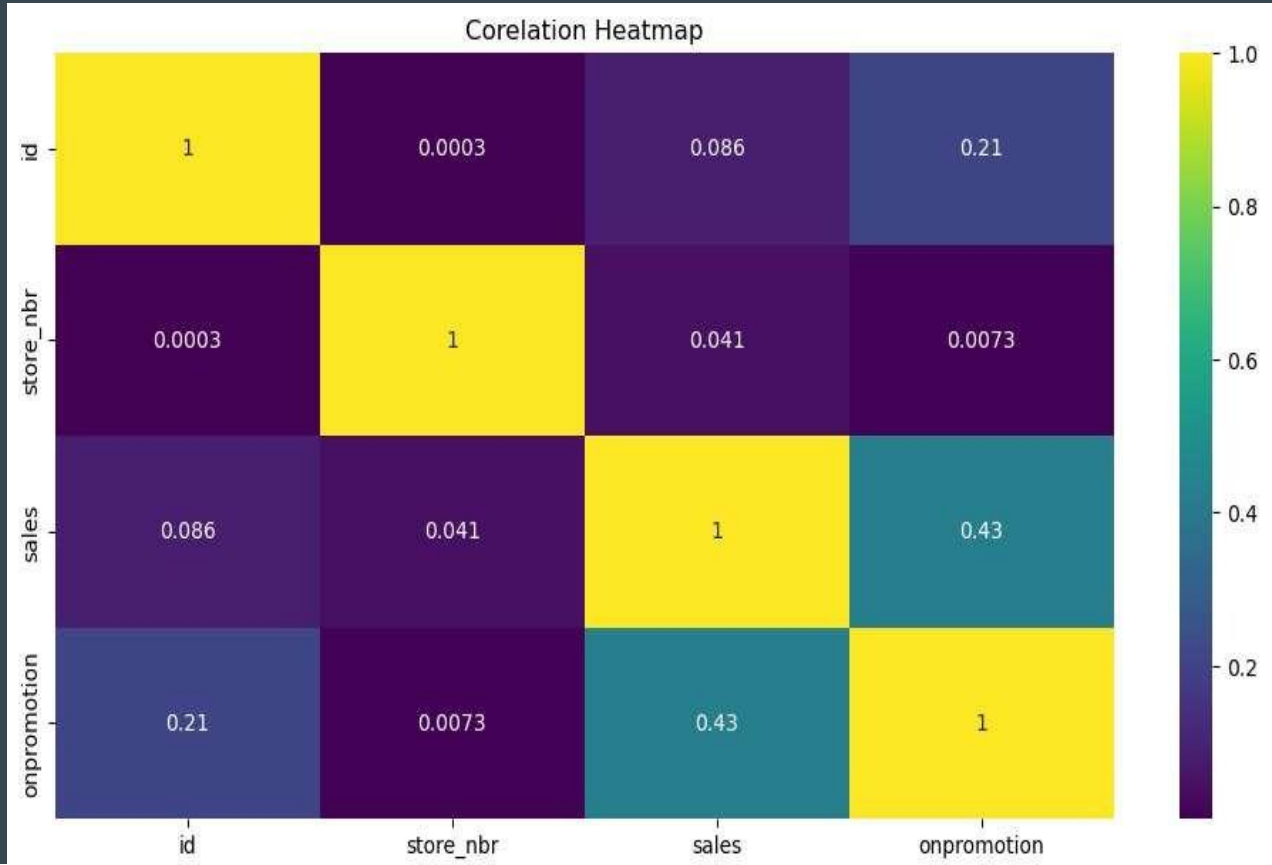


Distribution of sales based on family

- This the heatmap and heatmap is used to represent correlation .
- In this graph sales and promotion highly related to each other.
- So higher the promotion then higher we do the sales.

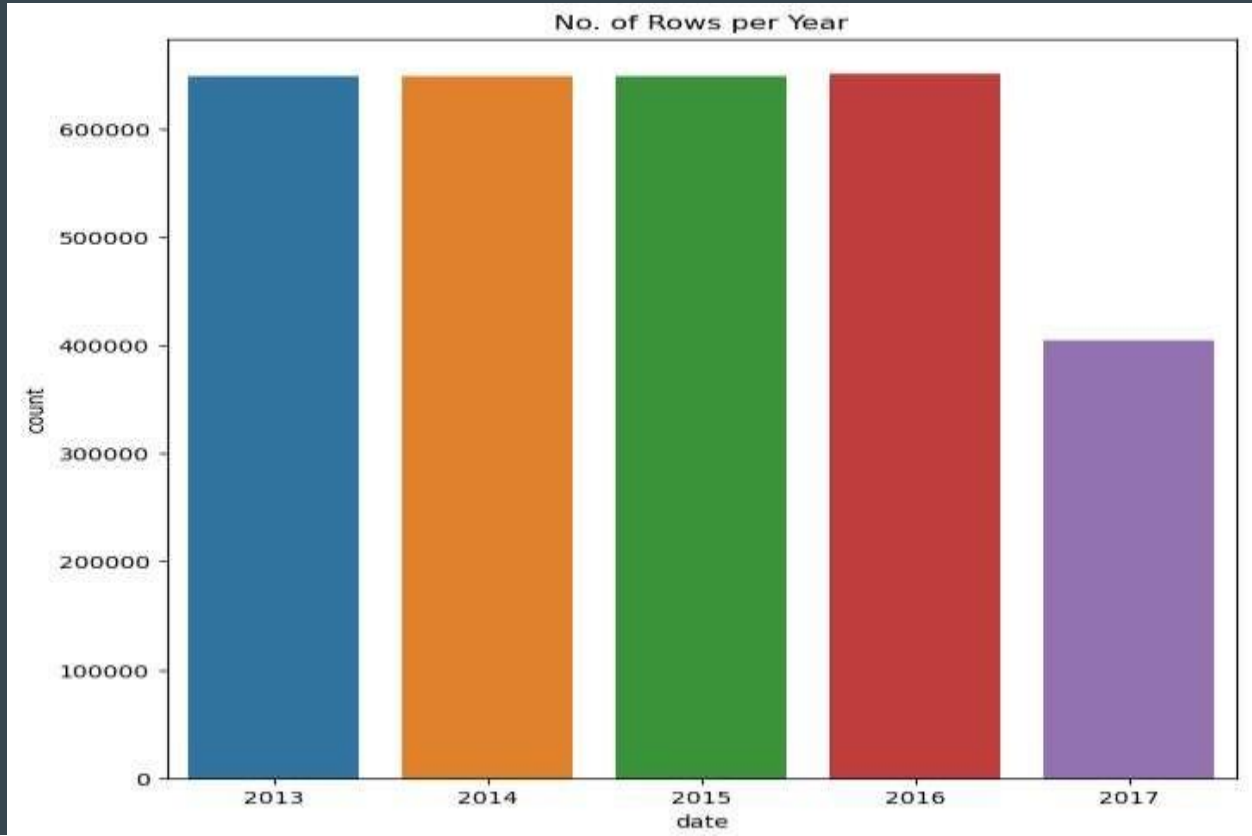
  Similarly there is no co relation between store number and id

  And yellow color is showing correlation with itself so is 1 than we ignore



Data available in each year: -

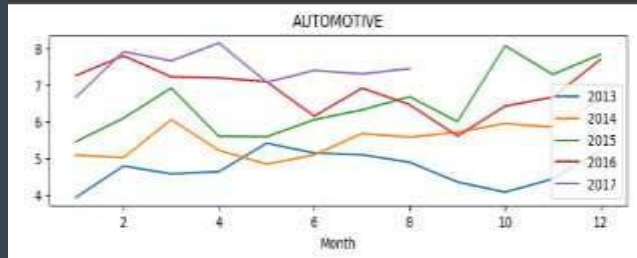
- This graph represents the data availability in each year.
- So in 2013,2014,2015,2016 almost the same data is available.
- In 2017 the less data is available.

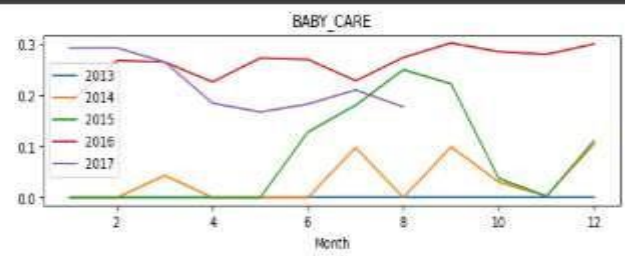


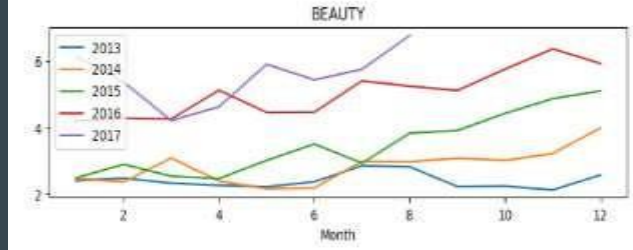
Product Family Monthly Sale: -

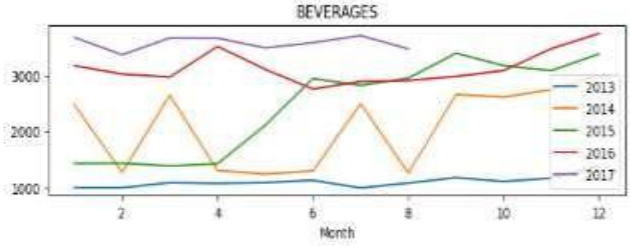
- It is a graph of month wise sales of the particular product. The blue line represents the month wise sales of particular product in 2013
- orange line represents the month wise sales of particular product in year 2014
- green line represents the year 2015,
- red line represents the year 2016
- purple line represents 2017.

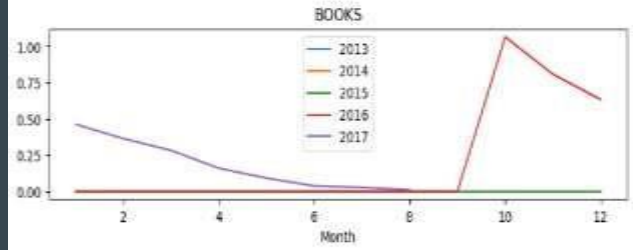
- And we can conclude from the first graph the monthly sales of the automotive product in 2017 is high.

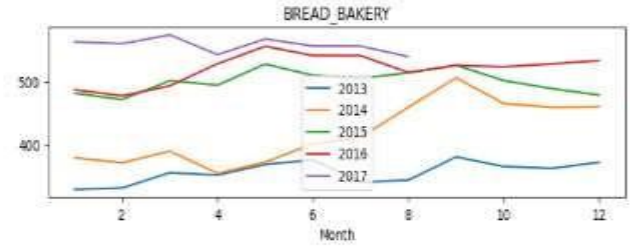


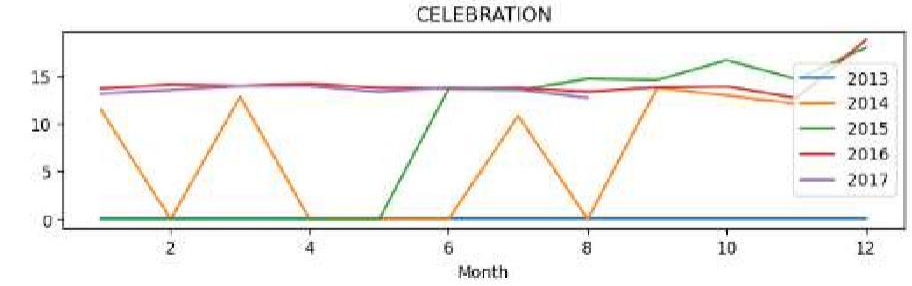


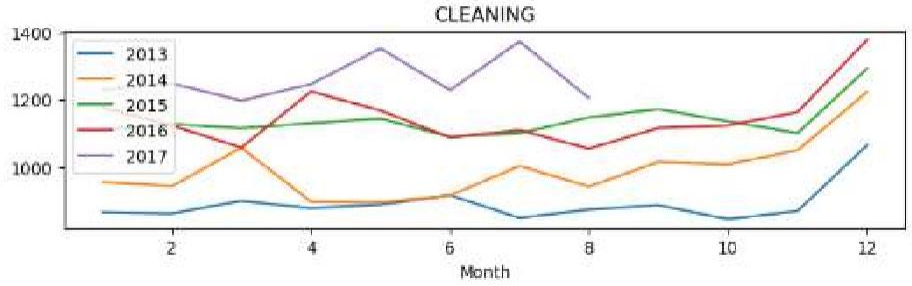


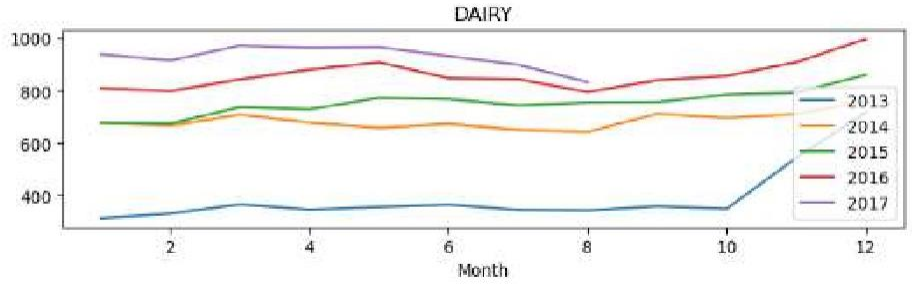


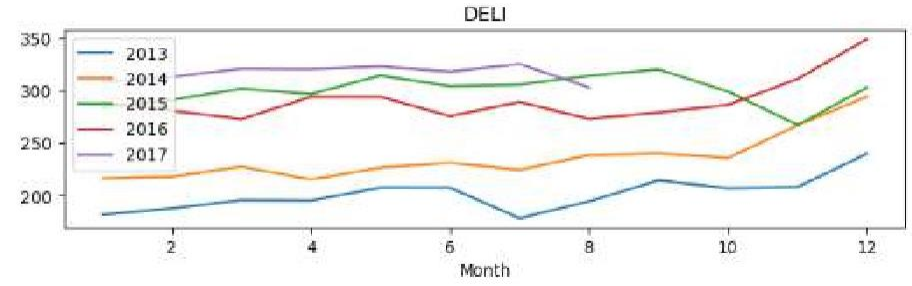


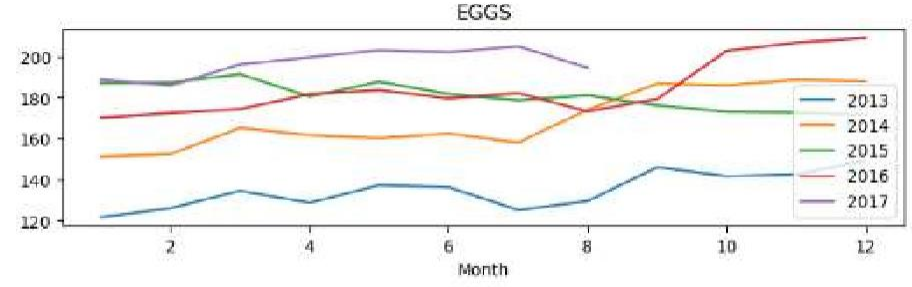


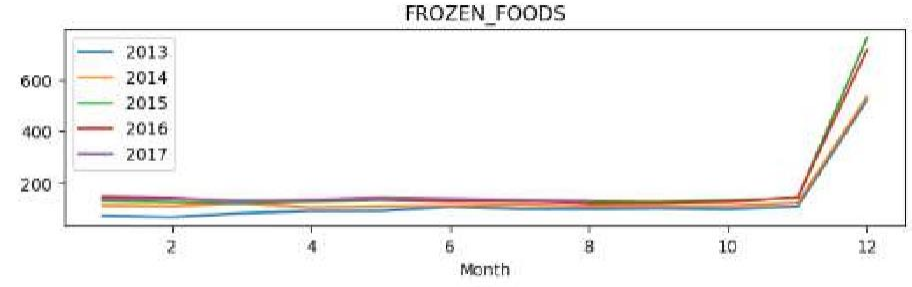

CELEBRATION
Month
2013
2014
2015
2016
2017


CLEANING
Month
2013
2014
2015
2016
2017

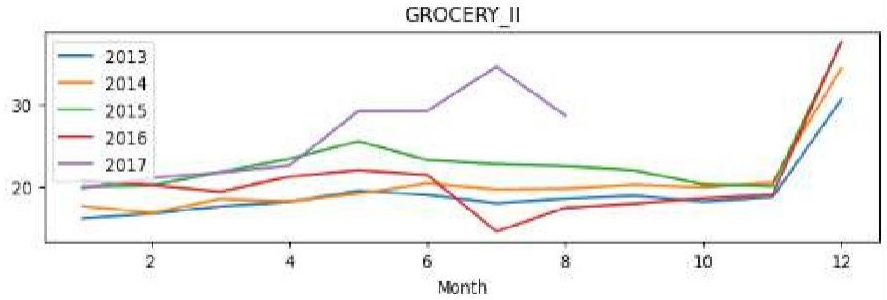

DAIRY
Month
2013
2014
2015
2016
2017

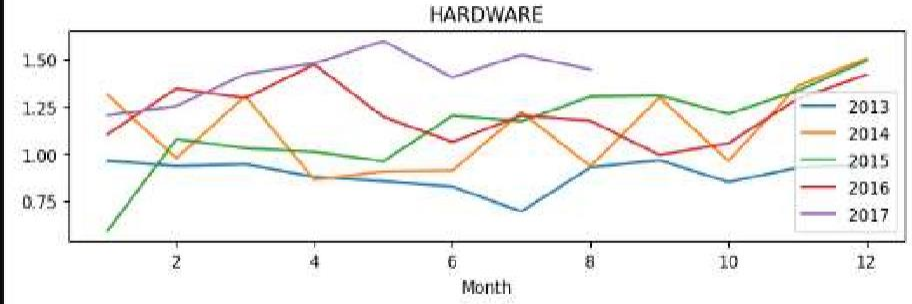

DELI
Month
2013
2014
2015
2016
2017


EGGS
Month
2013
2014
2015
2016
2017

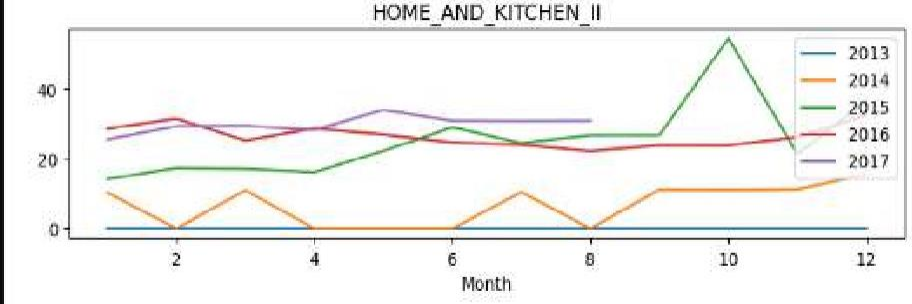

FROZEN_FOODS
Month
2013
2014
2015
2016
2017

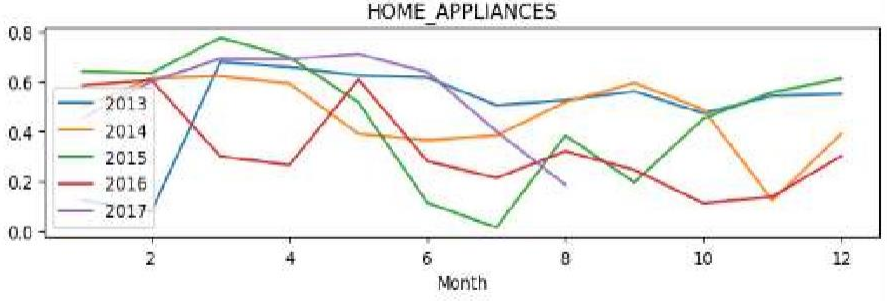

GROCERY_I
2013
2014
2015
2016
2017
Month

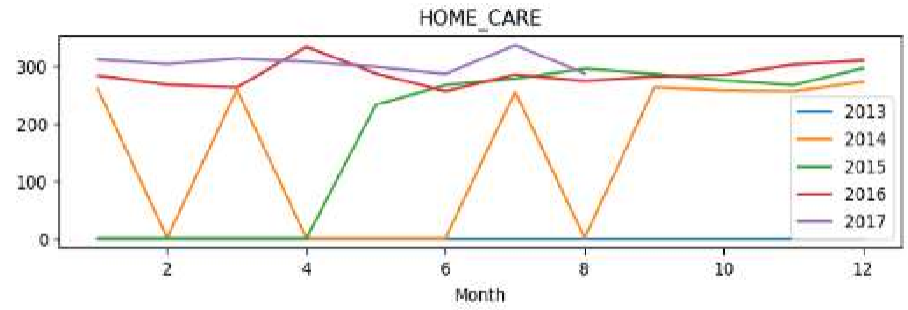

GROCERY_II
2013
2014
2015
2016
2017
Month


HARDWARE
2013
2014
2015
2016
2017
Month

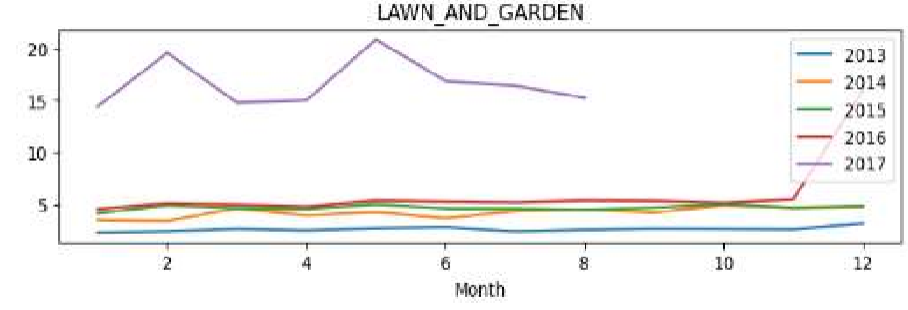

HOME_AND_KITCHEN_I
2013
2014
2015
2016
2017
Month

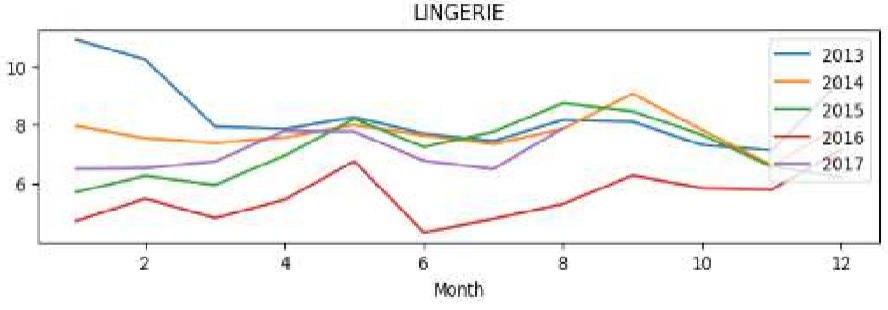

HOME_AND_KITCHEN_II
2013
2014
2015
2016
2017
Month

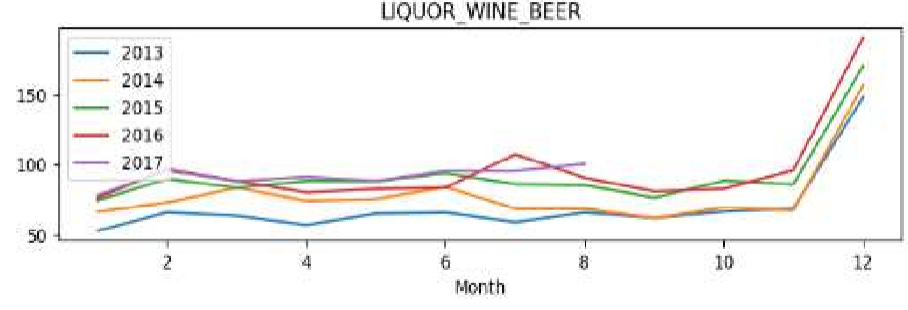

HOME_APPLIANCES
2013
2014
2015
2016
2017
Month

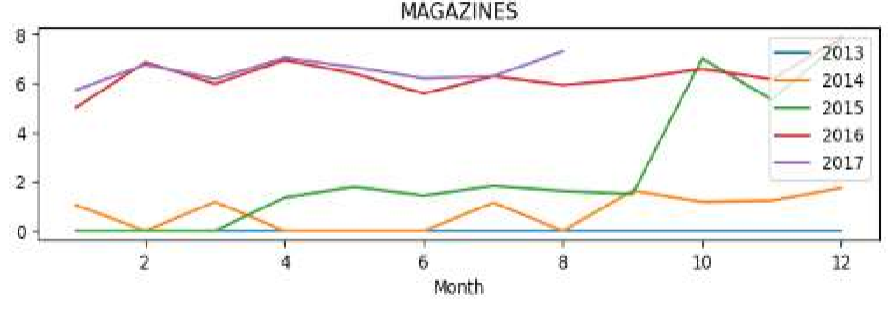

HOME_CARE
300
200
100
0
2
4
6
8
10
12
Month
2013
2014
2015
2016
2017

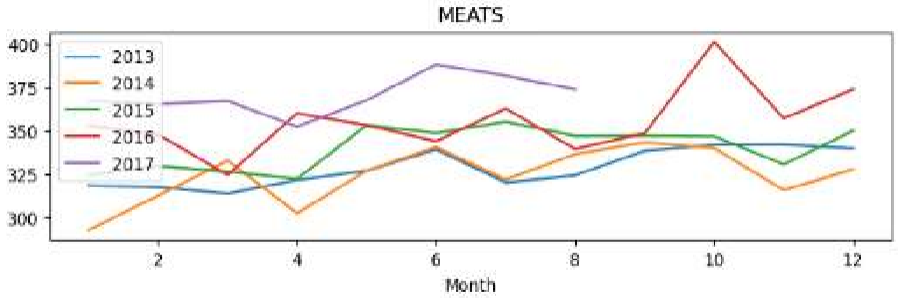

LADIESWEAR
10
5
0
2
4
6
8
10
12
Month
2013
2014
2015
2016
2017

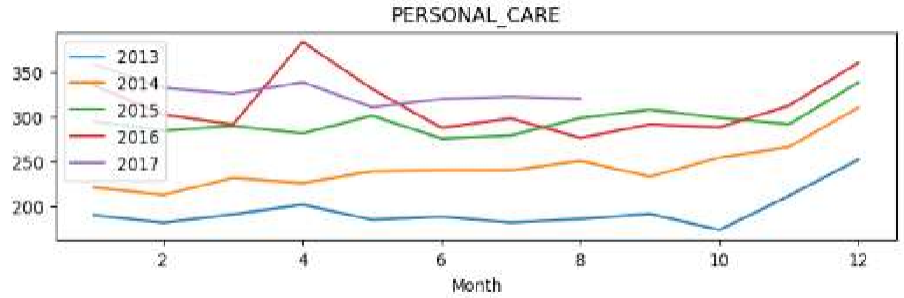

LAWN_AND_GARDEN
20
15
10
5
2
4
6
8
10
12
Month
2013
2014
2015
2016
2017

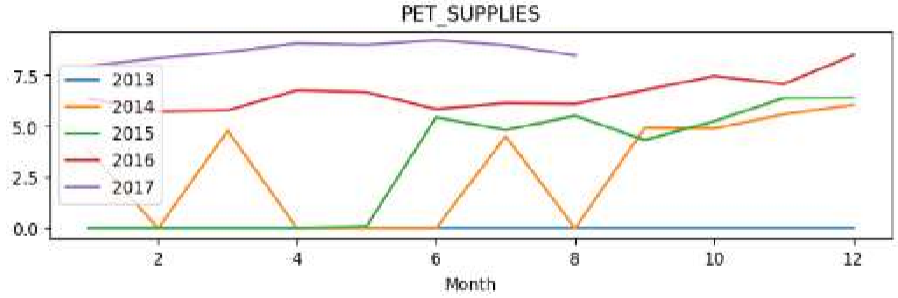

LINGERIE
10
8
6
2
4
6
8
10
12
Month
2013
2014
2015
2016
2017

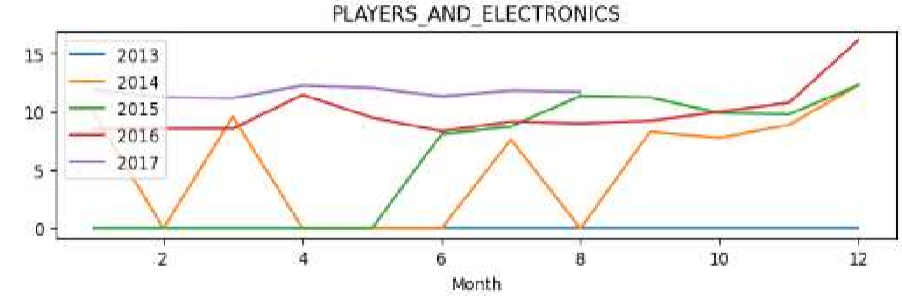

LIQUOR_WINE_BEER
150
100
50
2
4
6
8
10
12
Month
2013
2014
2015
2016
2017

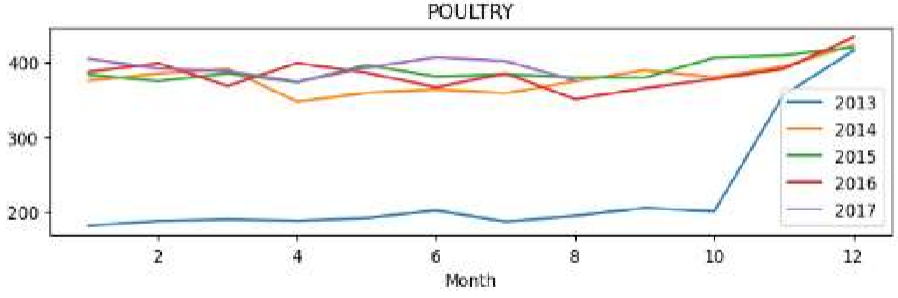

MAGAZINES
8
6
4
2
0
2
4
6
8
10
12
Month
2013
2014
2015
2016
2017

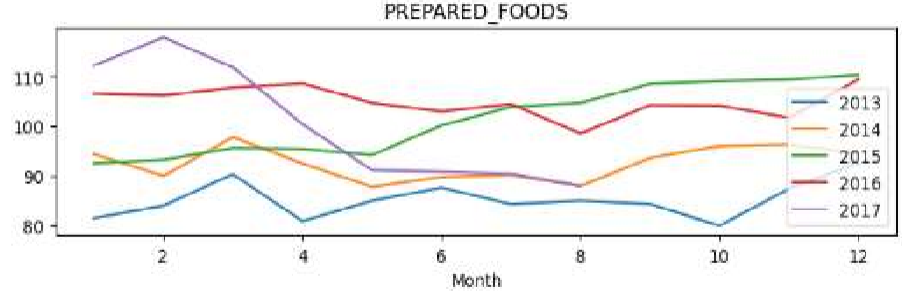

MEATS
2013
2014
2015
2016
2017
Month

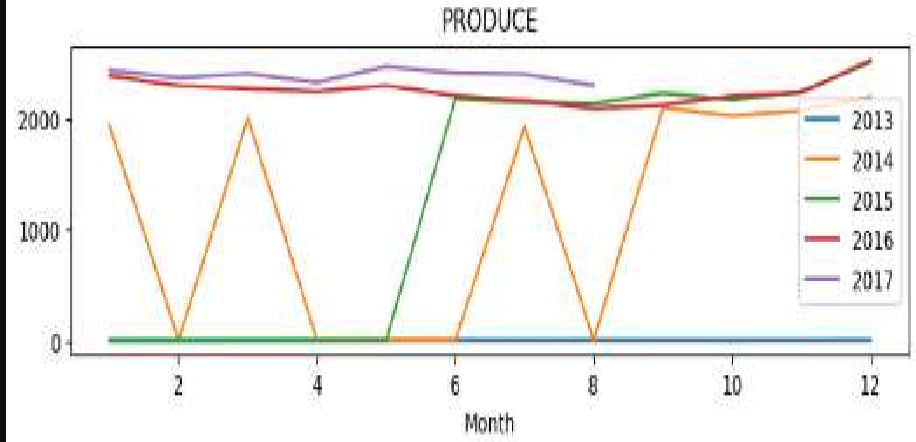

PERSONAL_CARE
2013
2014
2015
2016
2017
Month

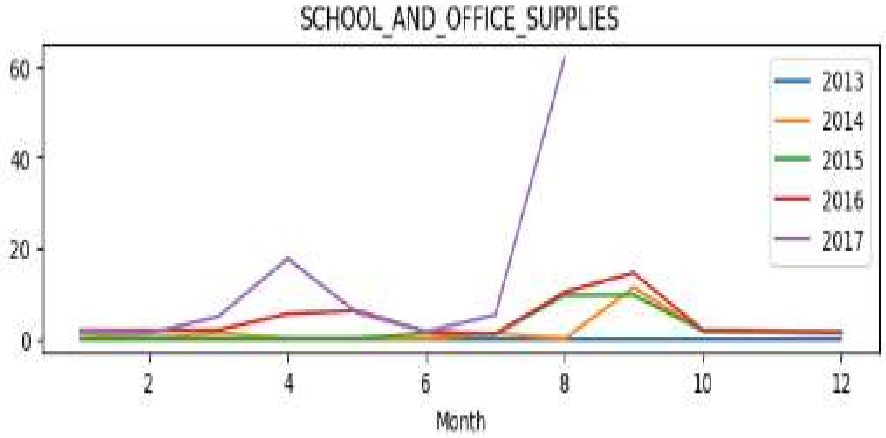

PET_SUPPLIES
2013
2014
2015
2016
2017
Month

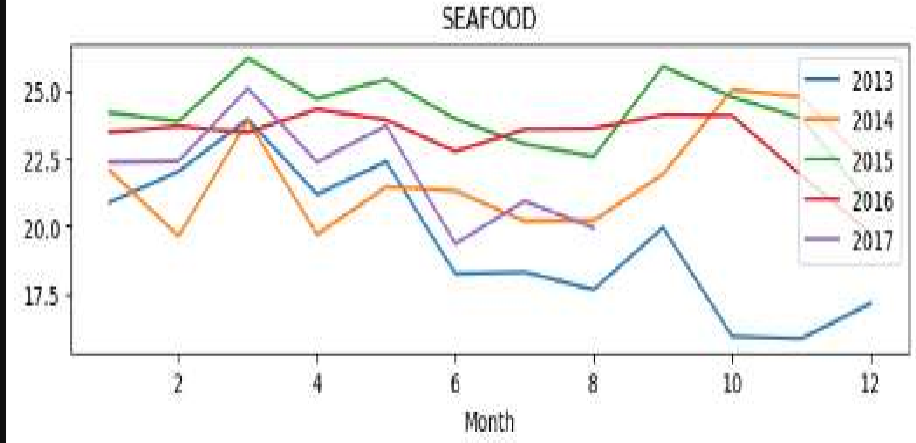

PLAYERS_AND_ELECTRONICS
2013
2014
2015
2016
2017
Month


POULTRY
2013
2014
2015
2016
2017
Month


PREPARED_FOODS
2013
2014
2015
2016
2017
Month

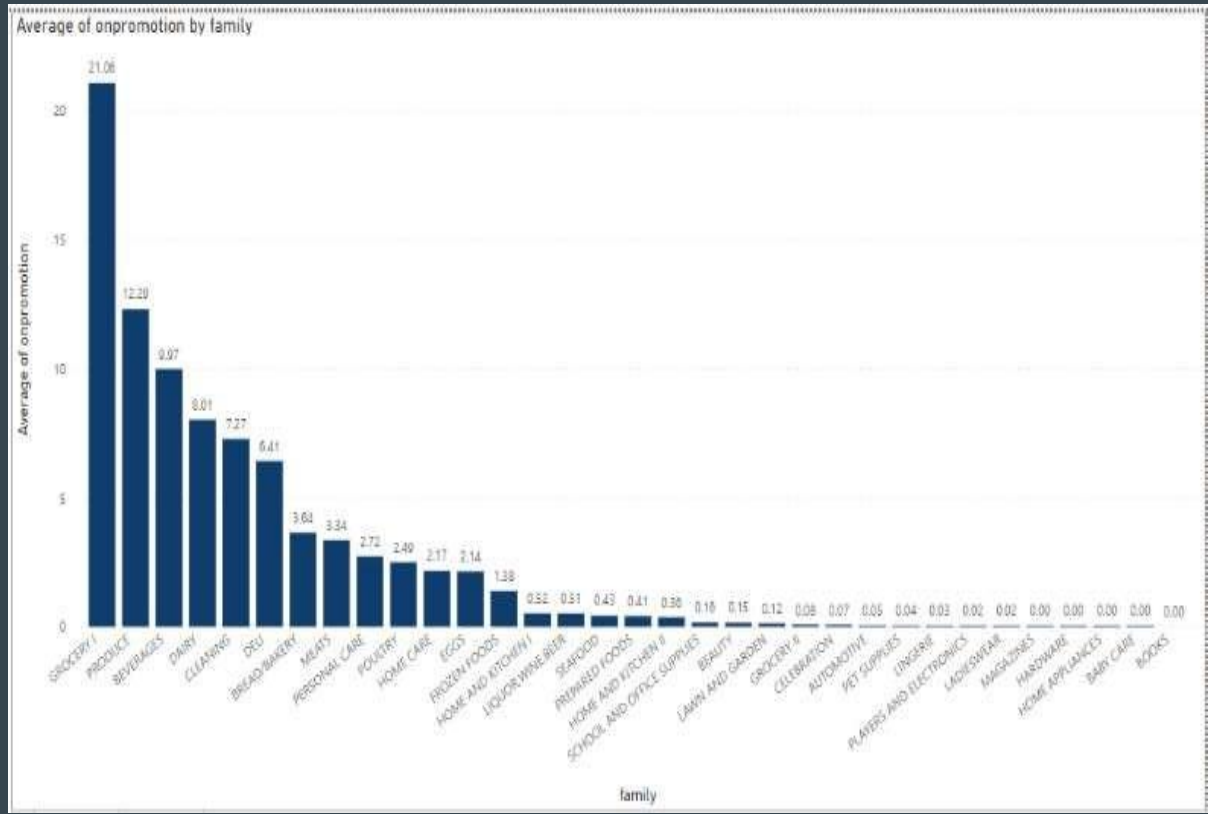

PRODUCE
2000
1000
0
2
4
6
8
10
12
Month
2013
2014
2015
2016
2017


SCHOOL_AND_OFFICE_SUPPLIES
60
40
20
0
2
4
6
8
10
12
Month
2013
2014
2015
2016
2017


SEAFOOD
25.0
22.5
20.0
17.5
2
4
6
8
10
12
Month
2013
2014
2015
2016
2017

Average onpromotion by Family : -

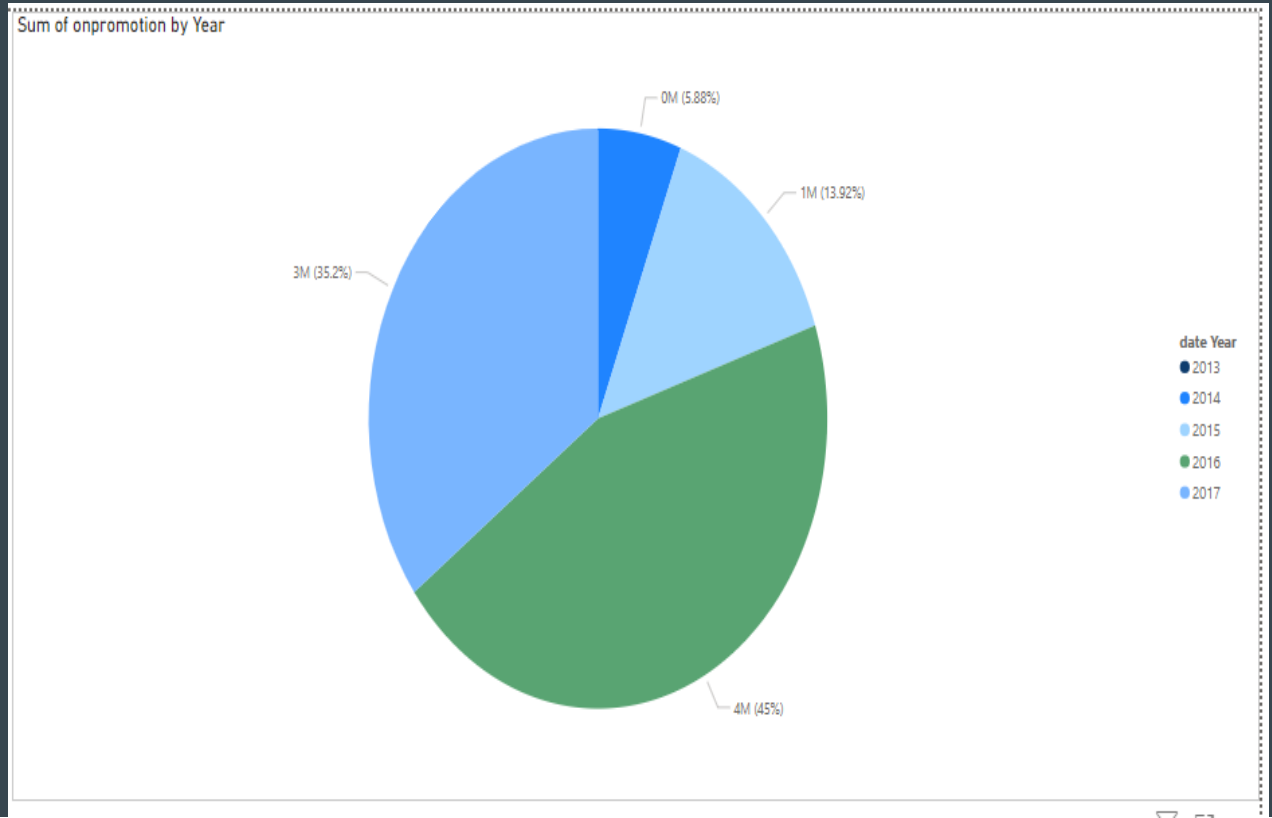
- Average Of onpromotion by family:- At 21.06, GROCERY I had the highest Average of on promotion and was Infinity higher than BOOKS.
- the lowest Average of onpromotion at 0.00.Across all 33 family,
- Average of onpromotion ranged from 0.00 to 21.06.

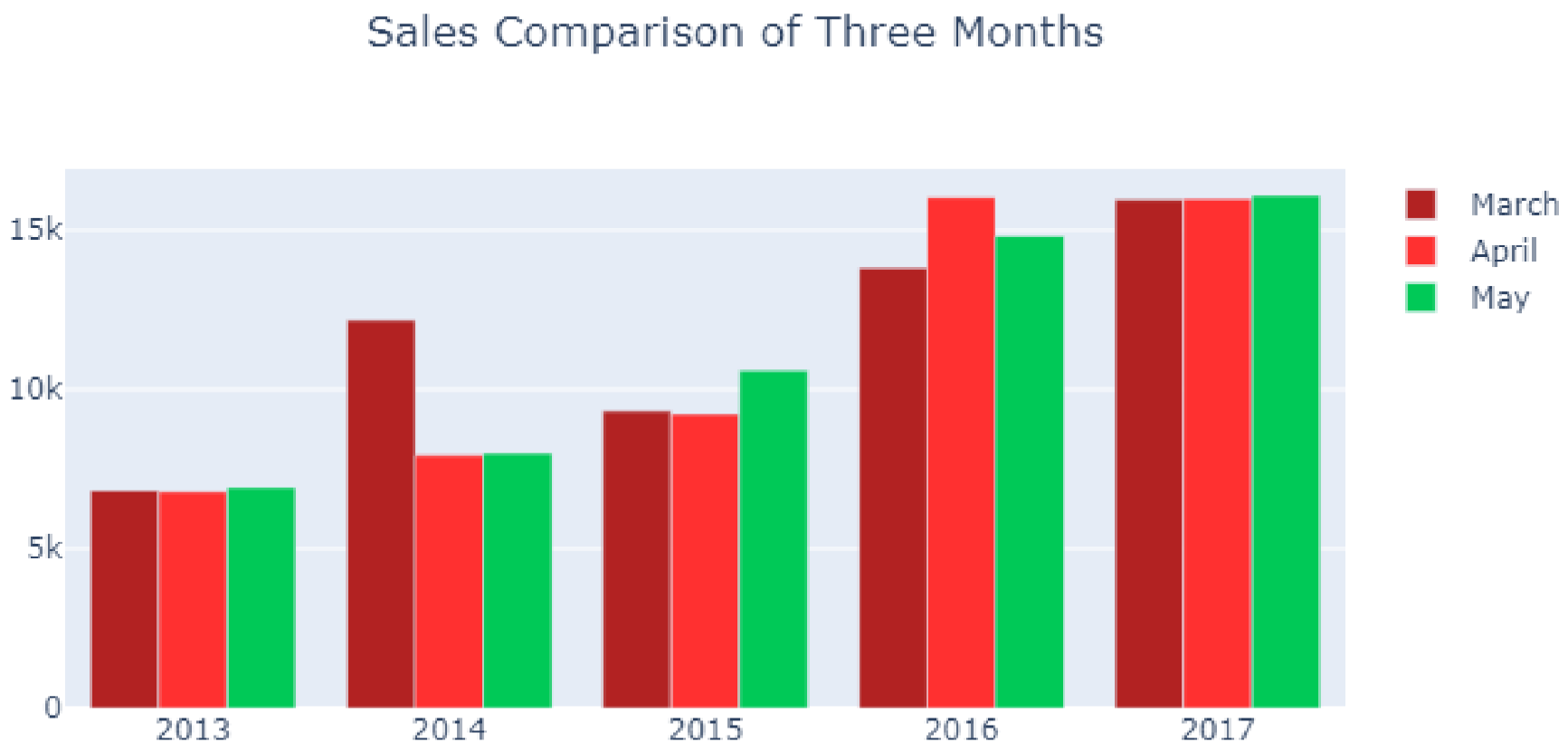


Sum of onpromotion by year

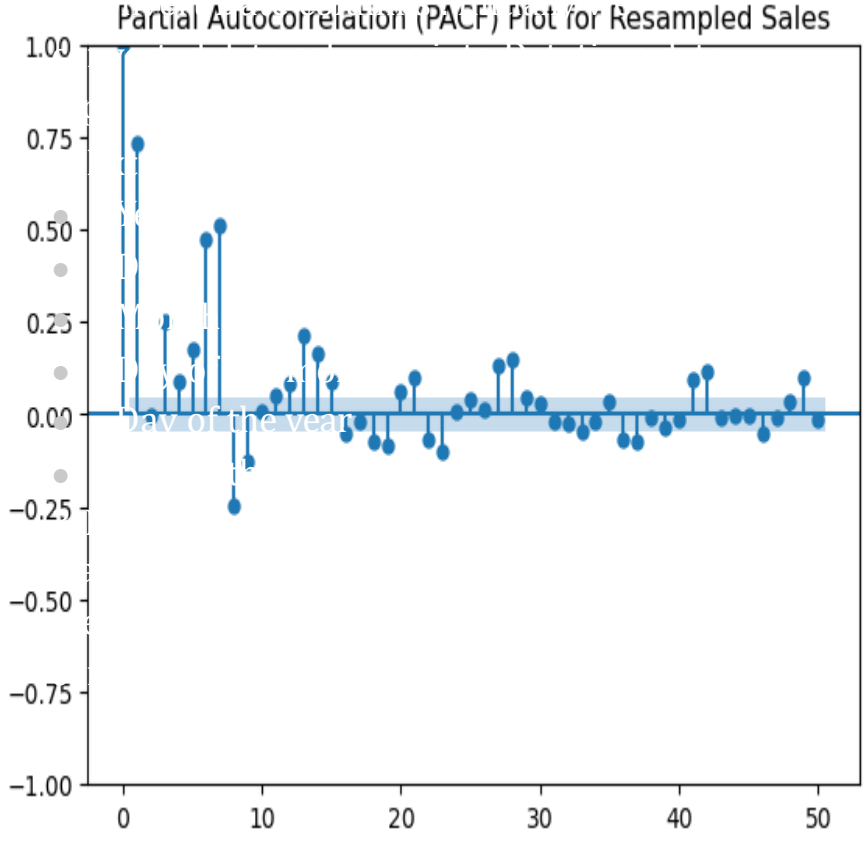
: -

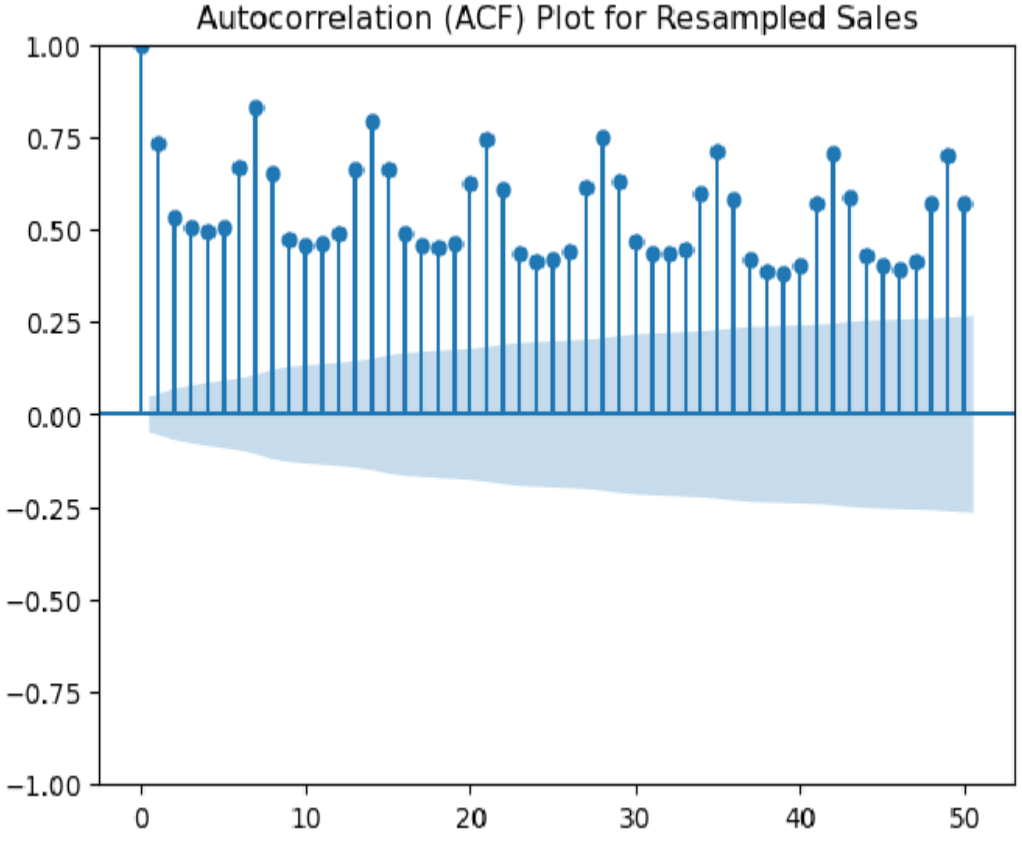
- It is a graph of year wise onpromotion. This graph tell us in 2013 there is no any promotion done on the products.
- And in 2014 there is a 5.88% of promotion on the products. Similarly in 2015,2016,2017 the promotion done as 13.92%,45%,35.2%. Highest promotion done in 2016 , and the lowest promotion done in a 2014.
- And the promotion on the products increases year by year.




Sales Comparison of Three Months
15k
10k
5k
0
2013
2014
2015
2016
2017
March
April
May

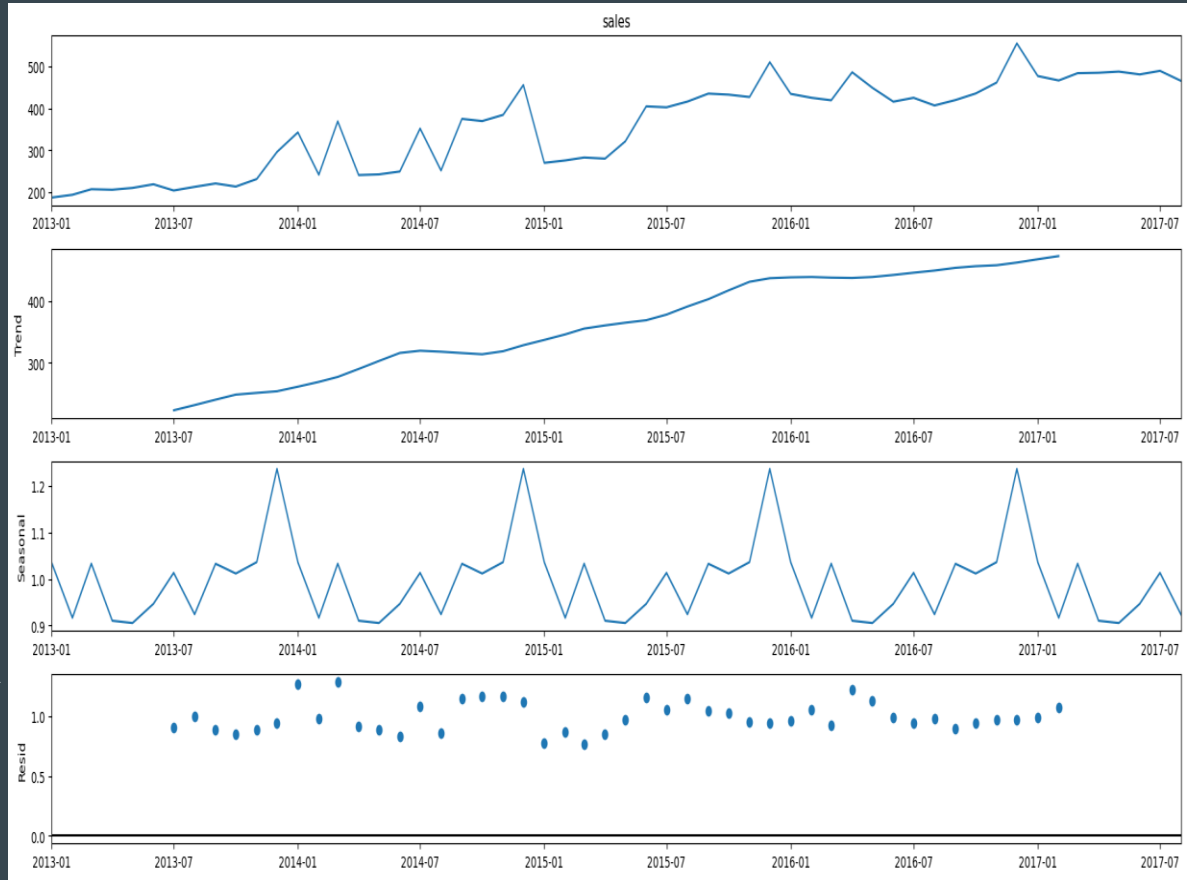
PACF AND ACF


Partial Autocorrelation (PACF) Plot for Resampled Sales
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
10
20
30
40
50


Autocorrelation (ACF) Plot for Resampled Sales
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
10
20
30
40
50

Seasonal Decomposition : Multiplicative

- In first graph is showing actual sales trend
- And second plot sowing the trend of data i.e. upward trend
- After that in third plot it is shoeing the seasonality of the data
- Then is showing the residuals of data
- So we find in seasonal decomposition multiplicative data have residuals zero and constant variance



Let’s Begin :-

Data

Pre Processing

Model

Building

Data Pre-Processing :-

1.Convert Date columns : Initially we converted date columns into Datetime data type

2. Extracted Feature :- week of the year

- Year
- Day
- Month
- Day of the month
- Day of the year
- Day of the week

3. Date Type Conversion: - the extracted feature are cast to appropriate data types for memory efficiency all extracted feature converted into unsigned integer type (unit 8 or uint16)

```
def create_date_features(df1):
    # Convert 'date' column to datetime type
    df1['date'] = pd.to_datetime(df1['date'])

    df1['WeekofYear'] = df1['date'].dt.isocalendar().week
    df1['Year'] = df1['date'].dt.year
    df1['Day'] = df1['date'].dt.day
    df1['Month'] = df1['date'].dt.month
    df1['day_of_month'] = df1['date'].dt.day
    df1['day_of_year'] = df1['date'].dt.dayofyear
    df1['day_of_week'] = df1['date'].dt.dayofweek

    df1['WeekofYear'] = df1['WeekofYear'].astype('uint8')
    df1['Month'] = df1['Month'].astype('uint8')
    df1['Year'] = df1['Year'].astype('uint16')
    df1['Day'] = df1['Day'].astype('uint8')
    df1['day_of_month'] = df1['day_of_month'].astype('uint16')
    df1['day_of_year'] = df1['day_of_year'].astype('uint16')
    df1['day_of_week'] = df1['day_of_week'].astype('uint16')
    return df1
```

Label Encoding

```
In [39]: df1.tail(5)
```

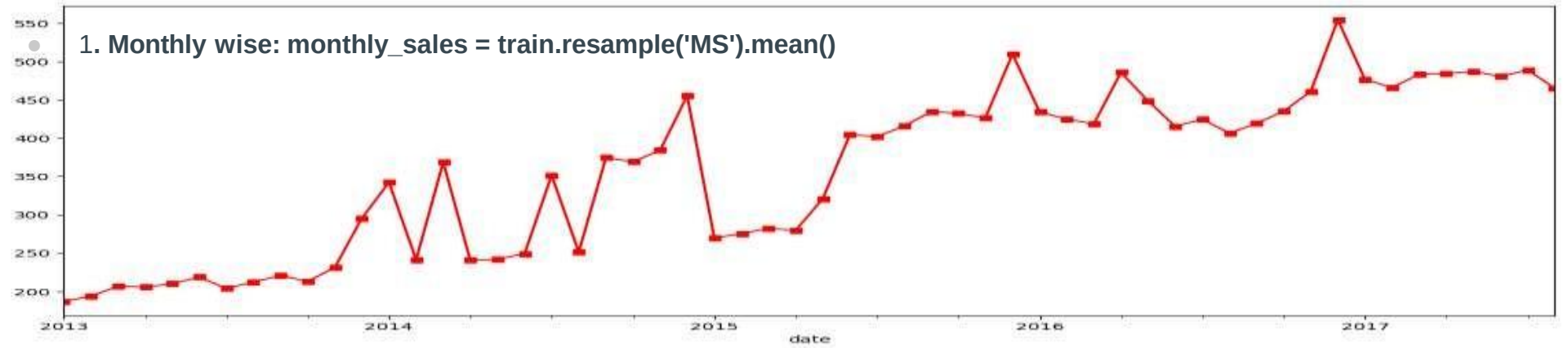
Out[39]:

	id	date	store_nbr	family	sales	onpromotion	WeekofYear	Year	Day	Month	day_of_month	day_of_year	day_of_week
3000883	3000883	2017-08-15	9	28	438.133	0	33	2017	15	8	15	227	1
3000884	3000884	2017-08-15	9	29	154.553	1	33	2017	15	8	15	227	1
3000885	3000885	2017-08-15	9	30	2419.729	148	33	2017	15	8	15	227	1
3000886	3000886	2017-08-15	9	31	121.000	8	33	2017	15	8	15	227	1
3000887	3000887	2017-08-15	9	32	16.000	0	33	2017	15	8	15	227	1

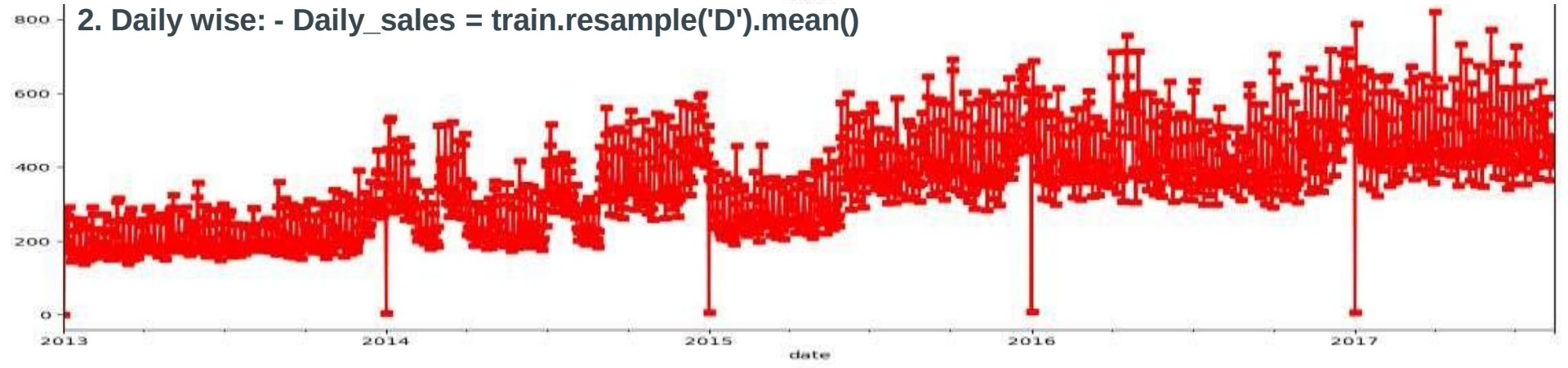
- In data we had one categorical data i.e product family.
- so for prediction and data processing we I converted into Label Encoding so after that data got changed form categorical into Numerical for analysis.
- Then I did furter step for building the model and analysis the behavior of data

Resampling data and Visualizing Sales Time Series Data

1. **Monthly wise: monthly_sales = train.resample('MS').mean()**



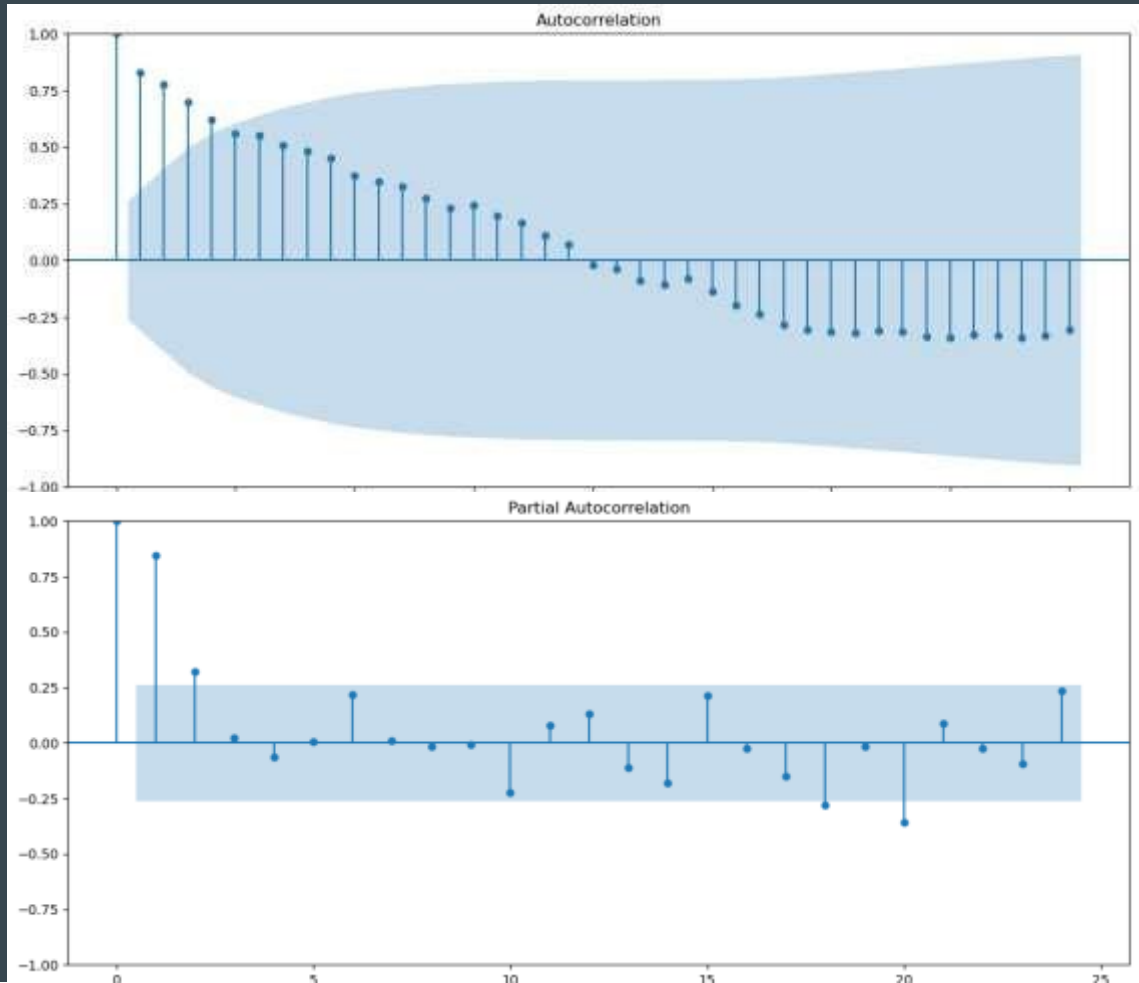
2. **Daily wise: - Daily_sales = train.resample('D').mean()**



ACF AND PACF PLOT

- import statsmodels.graphics.tsaplots as tsa_plots
- with plt.rc_context():
- plt.rc("figure", figsize=(14,6))
- tsa_plots.plot_acf(monthly_sales['sales'],lags=40)
- plt.show()

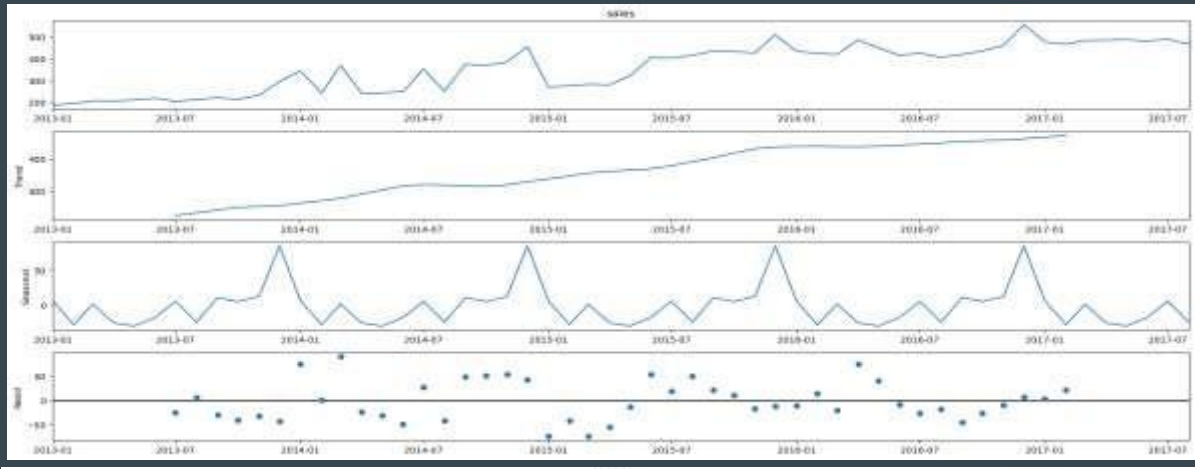
- import statsmodels.graphics.tsaplots as tsa_plots
- with plt.rc_context():
- plt.rc("figure", figsize=(14,6))
- tsa_plots.plot_pacf(monthly_sales['sales'],lags=24)
- plt.show()



Time Series Decomposition

Additive

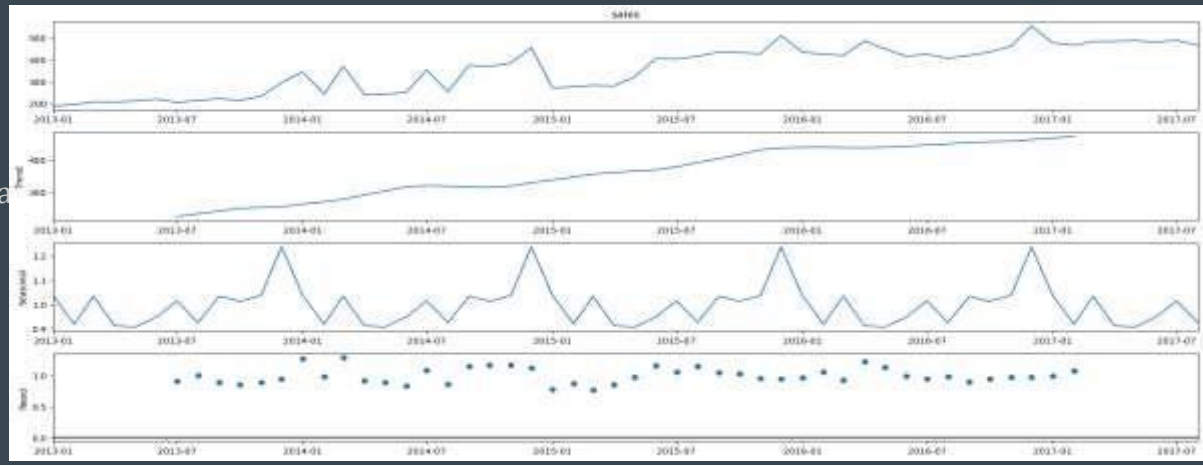
```
rcParams['figure.figsize'] = 18, 8
decomposition =
sm.tsa.seasonal_decompose(monthly_sales['sales'],
model='additive')
fig = decomposition.plot()
plt.show()
```



Multiplicative

rcParams['figure.figsize'] = 18, 8

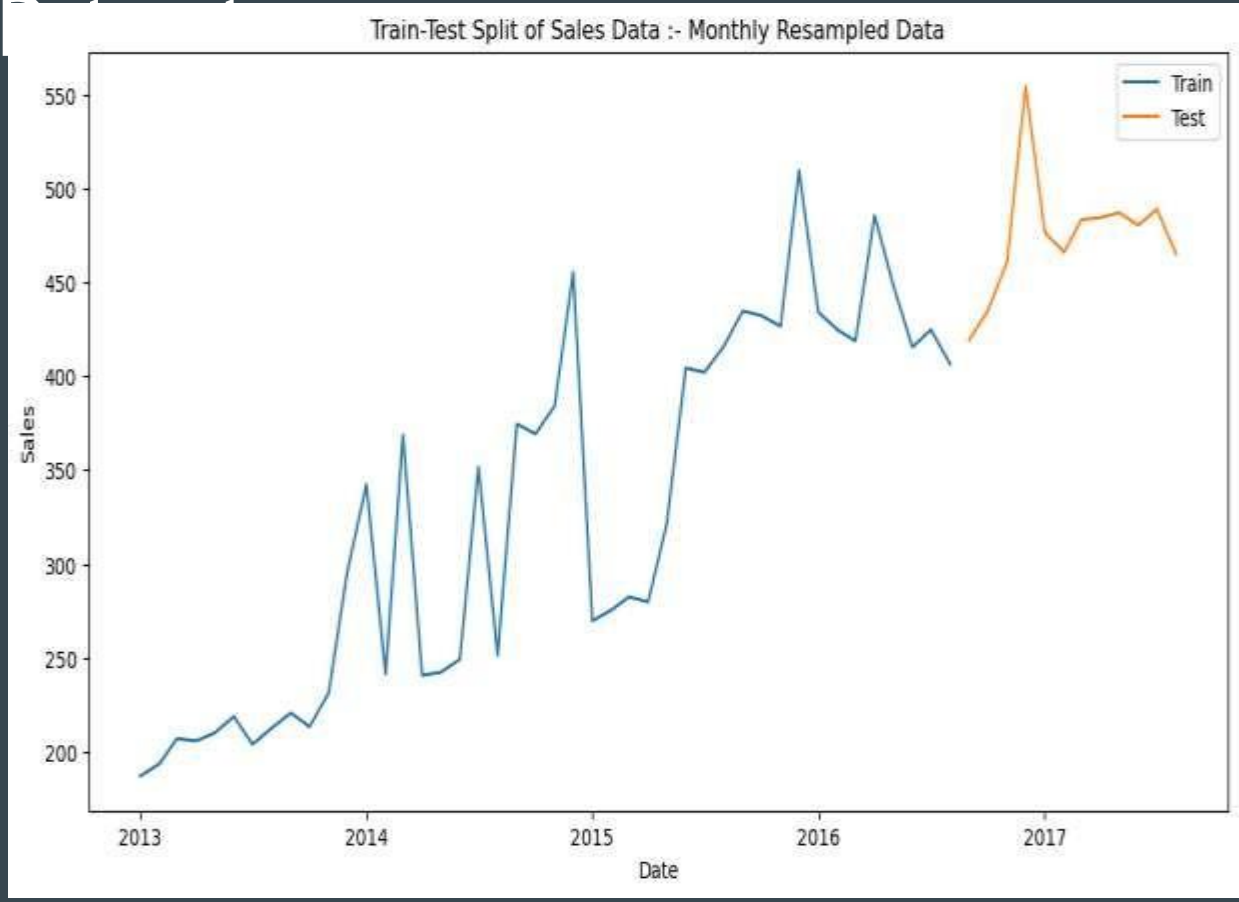
- decomposition = sm.tsa.seasonal_decompose(monthly_sales['s a es'], model='multiplicative')
- fig = decomposition.plot()
- plt.show()



Train Test Split Monthly

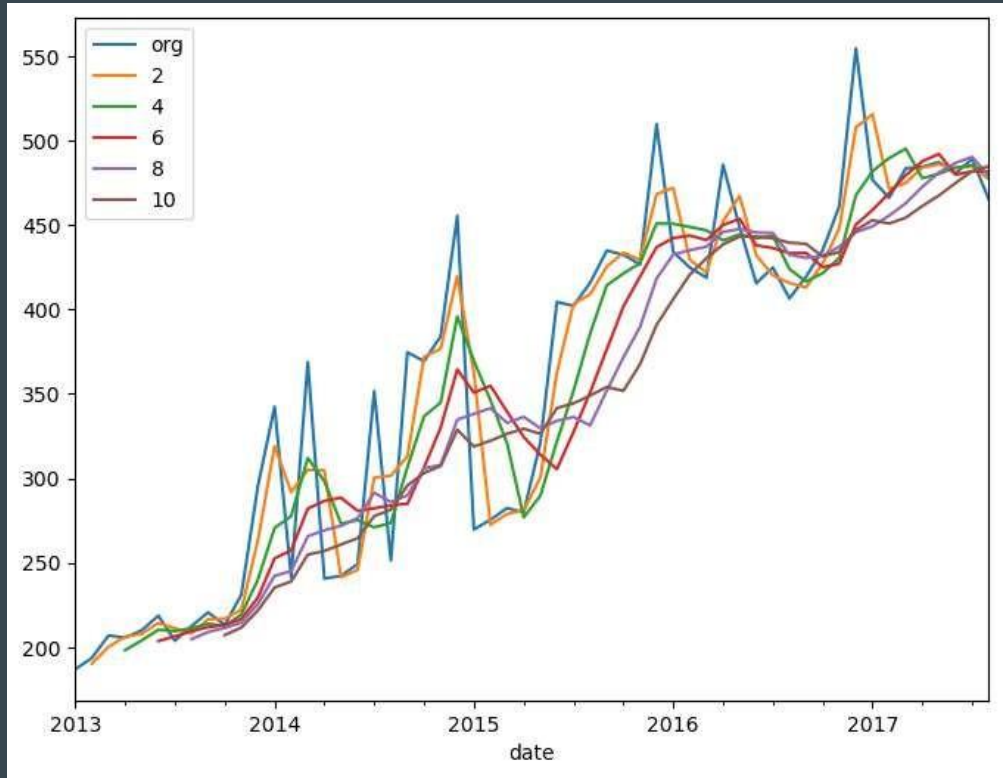
- from sklearn.model_selection import train_test_split
- size = 44
- training = monthly_sales[:size]
- testing = monthly_sales[size:]

- import matplotlib.pyplot as plt
- plt.figure(figsize=(12, 6))
- plt.plot(training.index, training['sales'], label='Train')
- plt.plot(testing.index, testing['sales'] ,label='Test')
- plt.xlabel('Date')
- plt.ylabel('Sales')
- plt.title('Train-Test Split of Sales Data :- Monthly Resampled Data')
- plt.legend()
- plt.show()



Moving Averages

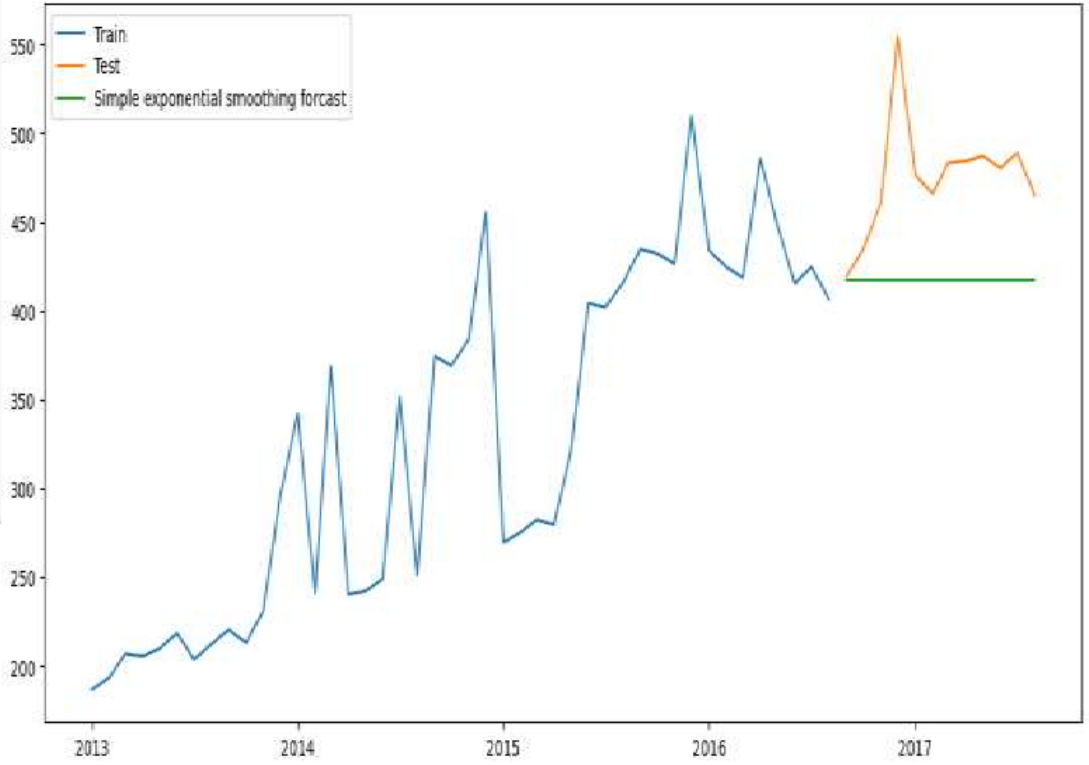
- plt.figure(figsize=(8,6))
- monthly_sales['sales'].plot(label="org")
- for i in range(2,12,2):
- monthly_sales['sales'].rolling(i).mean().plot(label=str(i))
- plt.legend(loc='best')



Simple Exponential Method

```
ses_model = SimpleExpSmoothing(training['sales']).fit(optimized=True)
test_pred_ses = ses_model.forecast(12)
train_pred_ses = ses_model.fittedvalues
test_rmse_ses = sqrt(mean_squared_error(test_pred_ses,testing['sales']))
train_rmse_ses = sqrt(mean_squared_error(train_pred_ses,training['sales']))
mape_ses = MAPE(test_pred_ses,testing['sales'])
print('Simple Exponential Method Evaluation',"\n",\
'Mean Absolute Percent Error   = {}'.format(mape_ses),"\n",\
'Train Root Mean Squared Error = {}'.format(train_rmse_ses),"\n",
'Test Root Mean Squared Error  = {}'.format(test_rmse_ses))
```

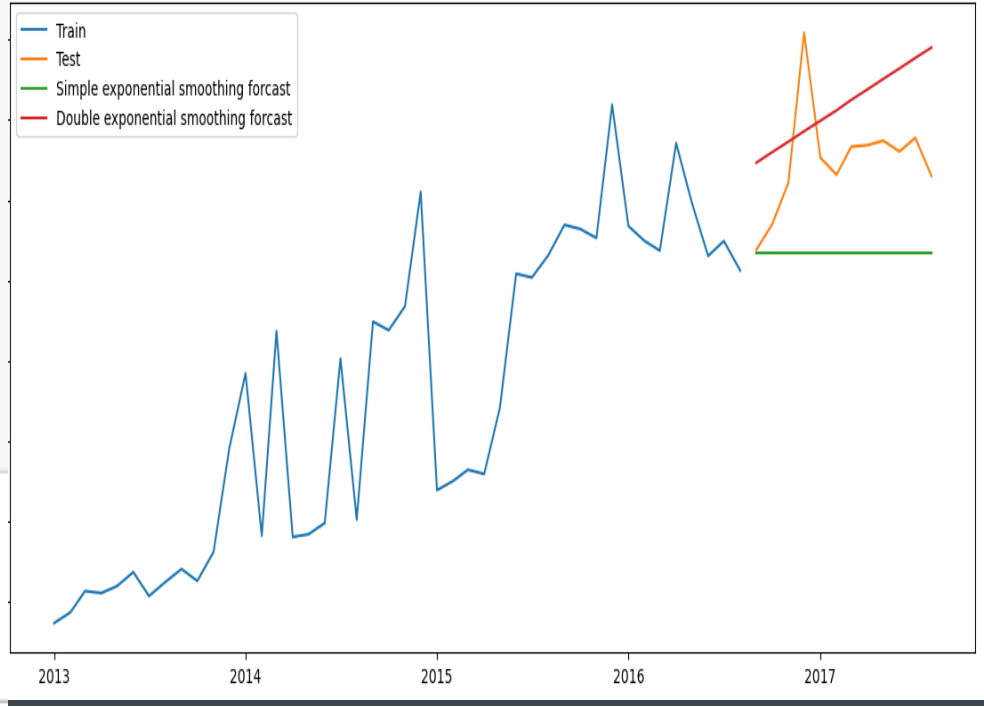
```
Simple Exponential Method Evaluation
 Mean Absolute Percent Error   = 11.719759966050413
 Train Root Mean Squared Error = 52.86011841553661
 Test Root Mean Squared Error  = 65.54528632903032
```



Holt method (Double Exponential)
Captures both Level and Trend

```
dexp_model  = Holt(training['sales']).fit(optimized=True)
test_pred_dexp = dexp_model.forecast(12)
train_pred_dexp = dexp_model.fittedvalues
test_rmse_dexp = sqrt(mean_squared_error(test_pred_dexp,testing['sales']))
train_rmse_dexp = sqrt(mean_squared_error(train_pred_dexp,training['sales']))
mape_dexp = MAPE(test_pred_dexp,training['sales'])
print('Simple Exponential Method Evaluation',"\n",\
'Mean Absolute Percent Error   = {}'.format(mape_dexp),"\n",\
'Train Root Mean Squared Error = {}'.format(train_rmse_dexp),"\n",
'Test Root Mean Squared Error  = {}'.format(test_rmse_dexp))
```

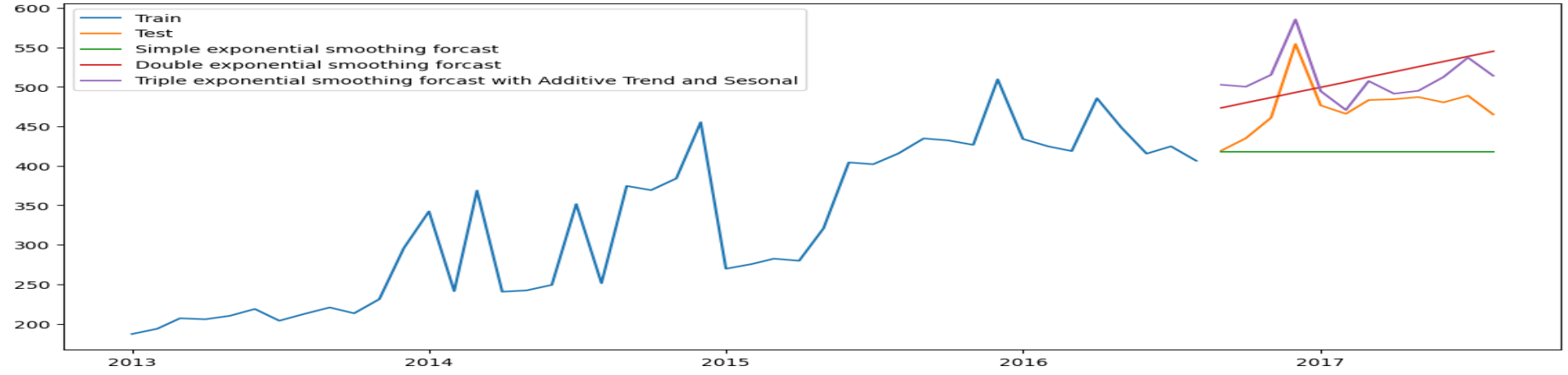
```
Simple Exponential Method Evaluation
 Mean Absolute Percent Error   = nan
 Train Root Mean Squared Error = 48.1928432586328
 Test Root Mean Squared Error  = 47.0742607846734
```



Holts winter exponential smoothing with additive seasonality and trend (Triple Exponential)

```
texp_add_model  = ExponentialSmoothing(training['sales'],seasonal="add",trend="add",seasonal_periods=12).fit(optimized=True)
test_pred_ad_texp = texp_add_model.forecast(12)
train_pred_ad_texp = texp_add_model.fittedvalues
test_rmse_ad_texp = sqrt(mean_squared_error(test_pred_ad_texp,testing['sales']))
train_rmse_ad_texp = sqrt(mean_squared_error(train_pred_ad_texp,training['sales']))
mape_ad_texp = MAPE(test_pred_ad_texp,testing['sales'])
print('Triple Exponential with Additive Trend and Seosonality Method Evaluation',"\n",\
'Mean Absolute Percent Error   = {}'.format(mape_ad_texp),"\n",\
'Train Root Mean Squared Error = {}'.format(train_rmse_ad_texp),"\n",
'Test Root Mean Squared Error  = {}'.format(test_rmse_ad_texp))
```

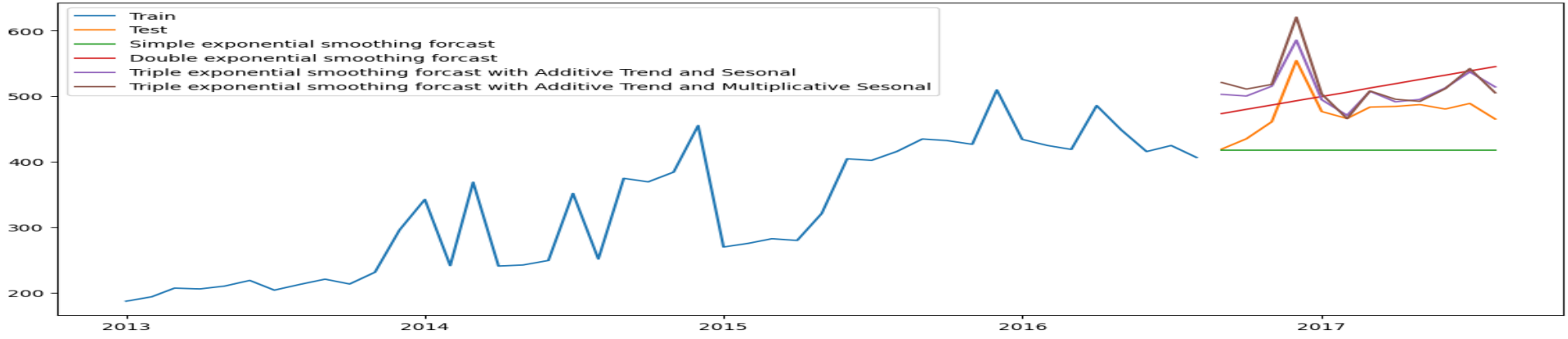
```
Triple Exponential with Additive Trend and Seosonality Method Evaluation
 Mean Absolute Percent Error   = 7.689319463331099
 Train Root Mean Squared Error = 39.626565915088
 Test Root Mean Squared Error  = 42.760071378109714
```



Holts winter exponential smoothing with multiplicative seasonality and additive trend (Triple Exponential)

```
texp_mul_ad_model  = ExponentialSmoothing(training['sales'],seasonal="mul",trend="add",seasonal_periods=12).fit(optimized=True)
test_pred_mul_ad_texp = texp_mul_ad_model.forecast(12)
train_pred_mul_ad_texp = texp_mul_ad_model.fittedvalues
test_rmse_mul_ad_texp = sqrt(mean_squared_error(test_pred_mul_ad_texp,testing['sales']))
train_rmse_mul_ad_texp = sqrt(mean_squared_error(train_pred_mul_ad_texp,training['sales']))
mape_mul_ad_texp = MAPE(test_pred_mul_ad_texp,testing['sales'])
print('Triple Exponential Method with Multiplicative Seasonality and Additive Trend',"\n",\
'Mean Absolute Percent Error   = {}'.format(mape_mul_ad_texp),"\n",\
'Train Root Mean Squared Error = {}'.format(train_rmse_mul_ad_texp),"\n",
'Test Root Mean Squared Error  = {}'.format(test_rmse_mul_ad_texp))
```

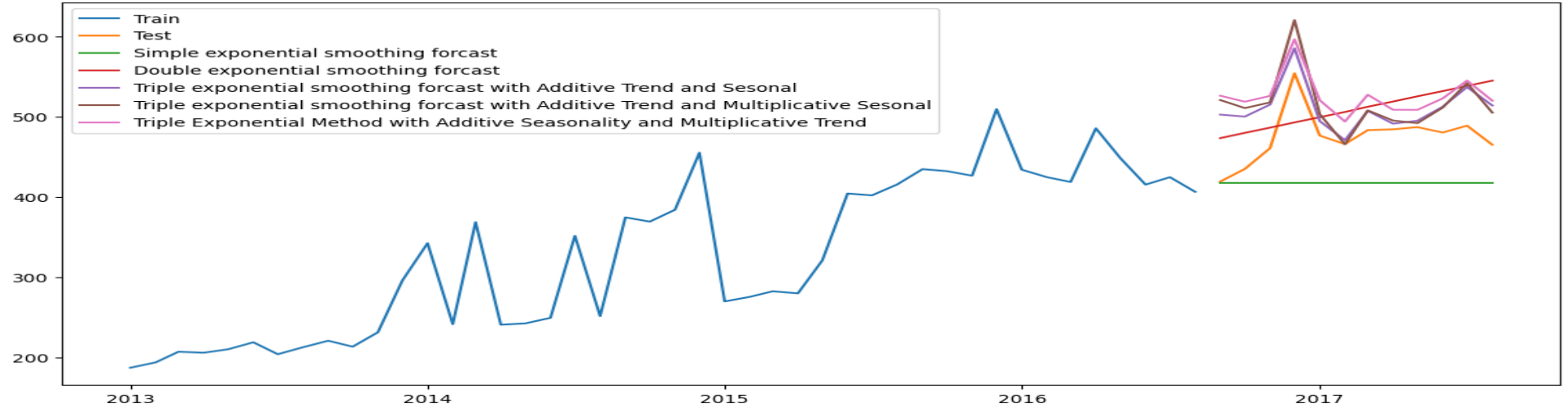
```
Triple Exponential Method with Multiplicative Seasonality and Additive Trend
 Mean Absolute Percent Error   = 8.835652068620005
 Train Root Mean Squared Error = 38.42626891591227
 Test Root Mean Squared Error  = 50.49665969582545
```



Holts winter exponential smoothing with Additive seasonality and multiplicative trend (Triple Exponential)¶

```
texp_ad_mul_model = ExponentialSmoothing(training['sales'],seasonal="add",trend="mul",
                    seasonal_periods=12).fit(smoothing_level=0.1, smoothing_slope=0.5)
test_pred_ad_mul_texp  = texp_ad_mul_model.forecast(12)
train_pred_ad_mul_texp  = texp_ad_mul_model.fittedvalues
test_rmse_ad_mul_texp  = sqrt(mean_squared_error(test_pred_ad_mul_texp ,testing['sales']))
train_rmse_ad_mul_texp  = sqrt(mean_squared_error(train_pred_ad_mul_texp ,training['sales']))
mape_ad_mul_texp  = MAPE(test_pred_ad_mul_texp,testing['sales'])
print('Triple Exponential Method with Additive Seasonality and Multiplicative Trend',"\n",\
'Mean Absolute Percent Error    = {}'.format(mape_ad_mul_texp ),"\n",\
'Train Root Mean Squared Error = {}'.format(train_rmse_ad_mul_texp ),"\n",
'Test Root Mean Squared Error  = {}'.format(test_rmse_ad_mul_texp ))
```

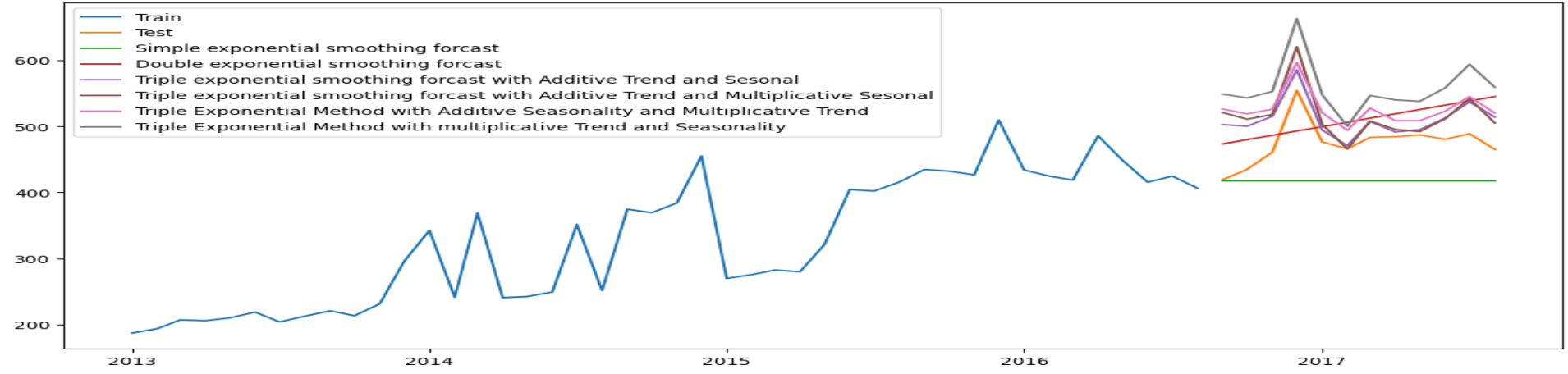
```
Triple Exponential Method with Additive Seasonality and Multiplicative Trend
 Mean Absolute Percent Error    = 11.058904074420822
 Train Root Mean Squared Error = 48.11043099427231
 Test Root Mean Squared Error  = 56.4940947533863
```



Holts winter exponential smoothing with multiplicative seasonality and multiplicative trend (Triple Exponential)

```
texp_mul_model  = ExponentialSmoothing(training['sales'],seasonal="mul",trend="mul",seasonal_periods=12).fit(optimized=True)
test_pred_mul_texp  = texp_mul_model.forecast(12)
train_pred_mul_texp  = texp_mul_model.fittedvalues
test_rmse_mul_texp  = sqrt(mean_squared_error(test_pred_mul_texp ,testing['sales']))
train_rmse_mul_texp  = sqrt(mean_squared_error(train_pred_mul_texp ,training['sales']))
mape_mul_texp  = MAPE(test_pred_mul_texp,testing['sales'])
print('Triple Exponential Method with multiplicative Trend and Seasonality',"\n",\
'Mean Absolute Percent Error   = {}'.format(mape_mul_texp ),"\n",\
'Train Root Mean Squared Error = {}'.format(train_rmse_mul_texp ),"\n",
'Test Root Mean Squared Error  = {}'.format(test_rmse_mul_texp ))
```

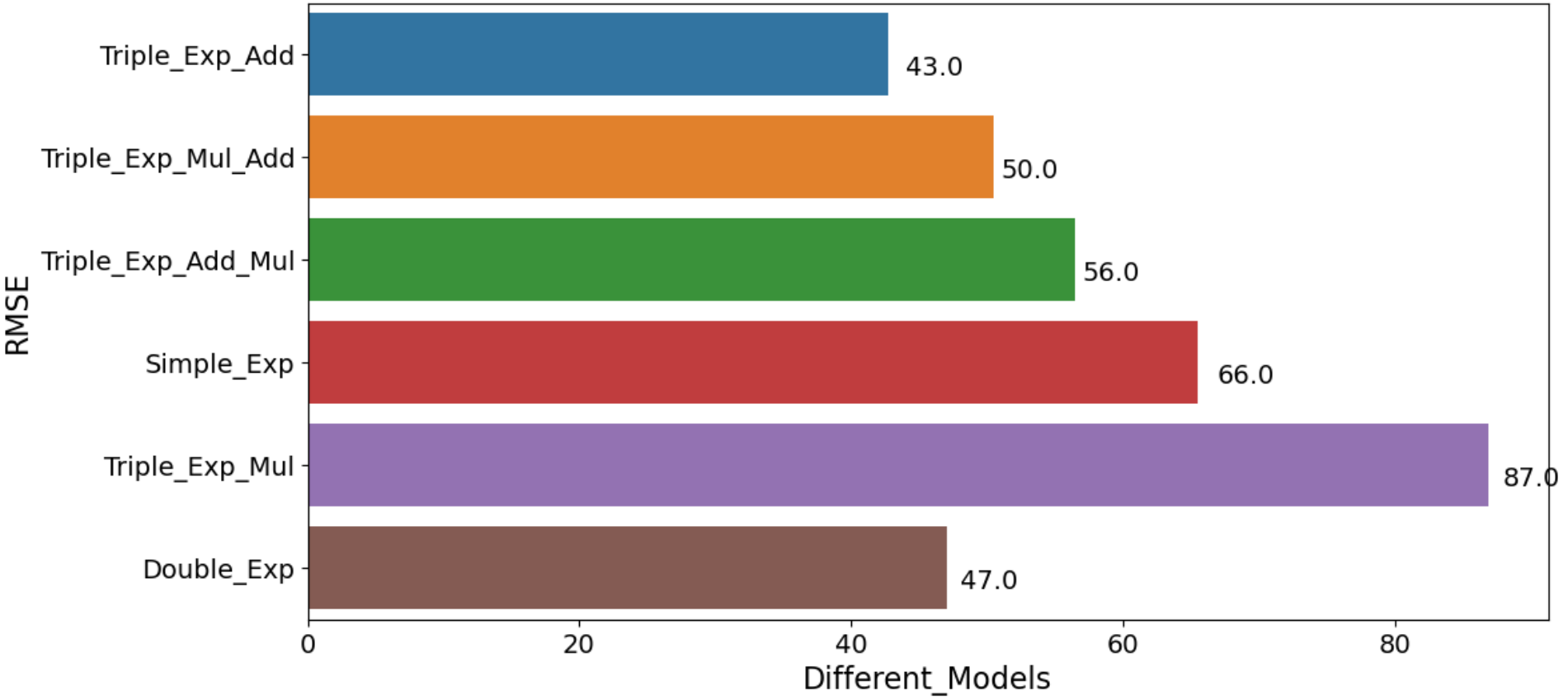
```
Triple Exponential Method with multiplicative Trend and Seasonality
 Mean Absolute Percent Error   = 17.55961832525947
 Train Root Mean Squared Error = 41.8581850377565
 Test Root Mean Squared Error  = 86.97009257180255
```

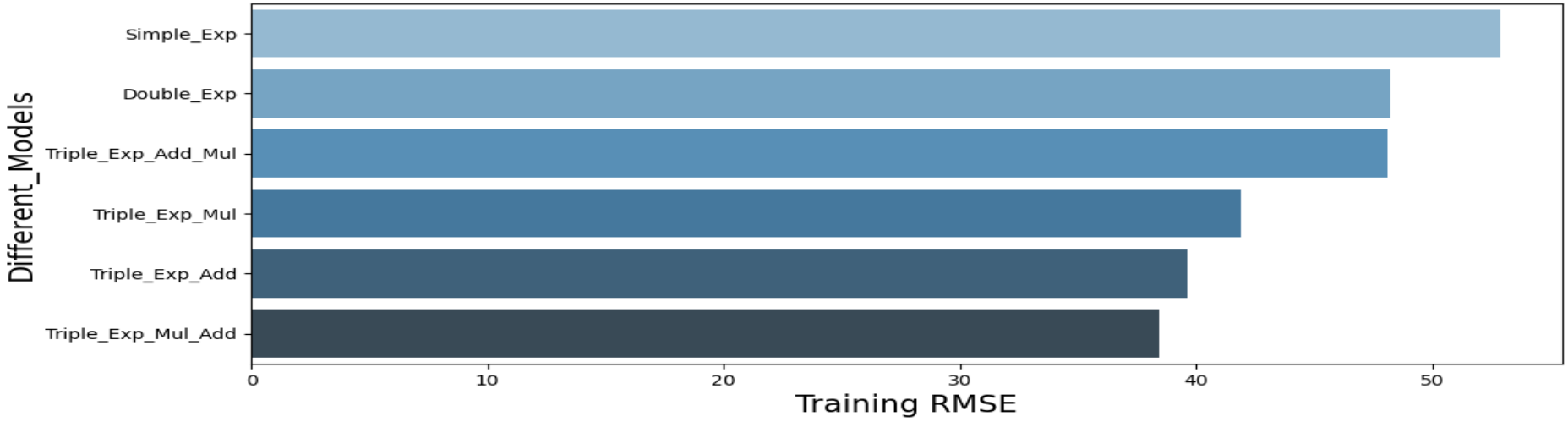


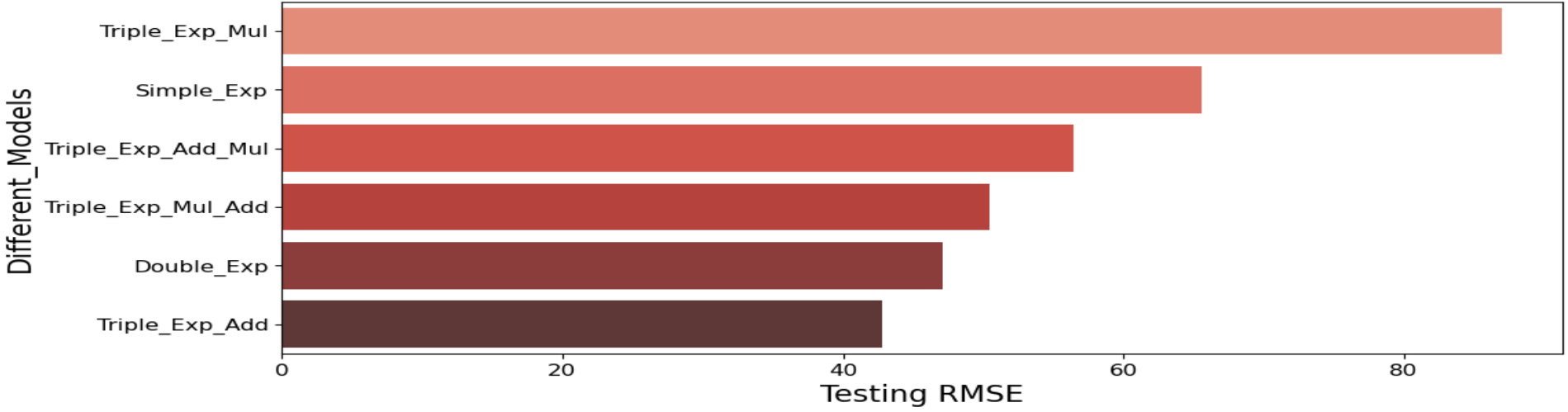
Comparing the results

	Models	Train_RMSE	Test_MAPE(%)	Test_RMSE_Values
0	Triple_Exp_Add	39.626566	7.689319	42.760071
1	Triple_Exp_Mul_Add	38.426269	8.835652	50.496660
2	Triple_Exp_Add_Mul	48.110431	11.058904	56.494095
3	Simple_Exp	52.860118	11.719760	65.545286
4	Triple_Exp_Mul	41.858185	17.559618	86.970093
5	Double_Exp	48.192843	NaN	47.074261

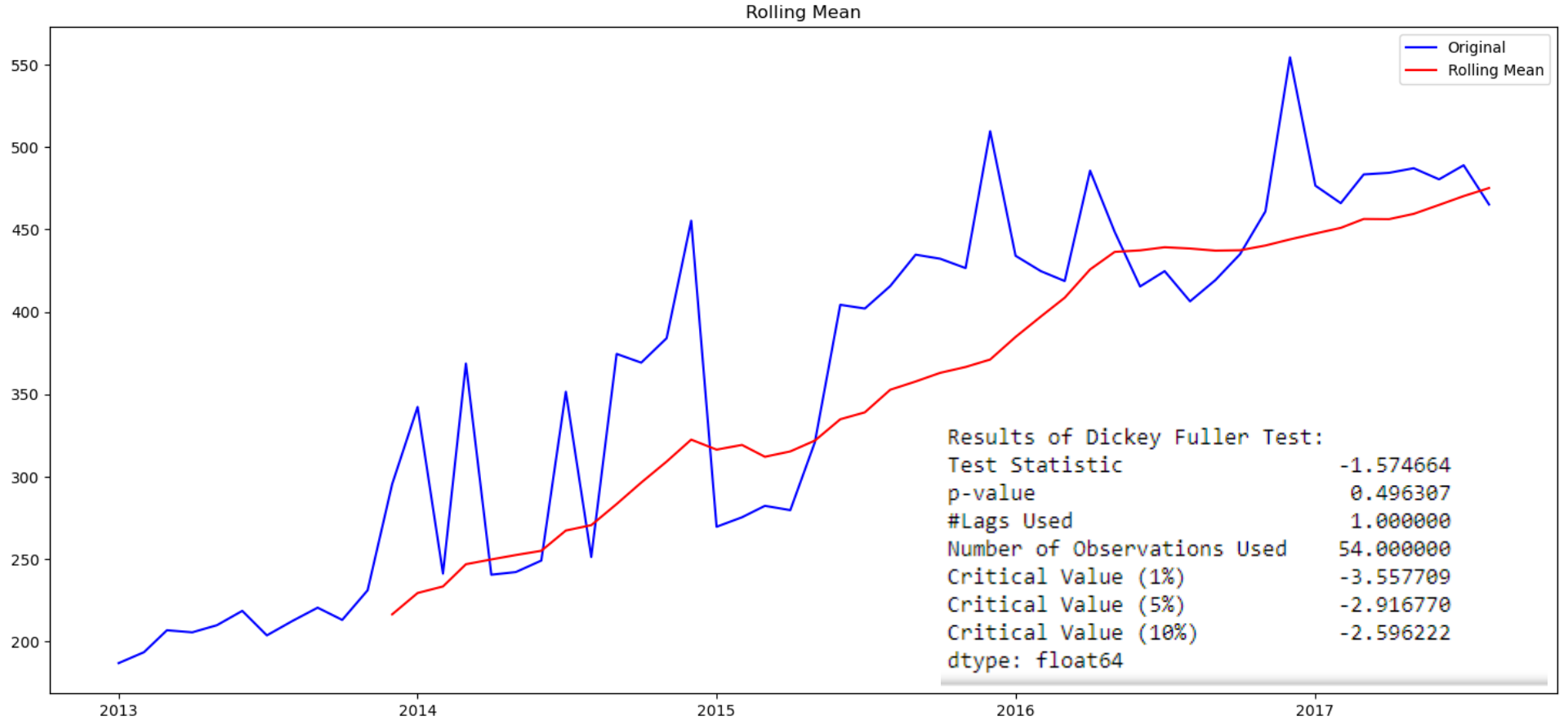
Visualizing Models Performance



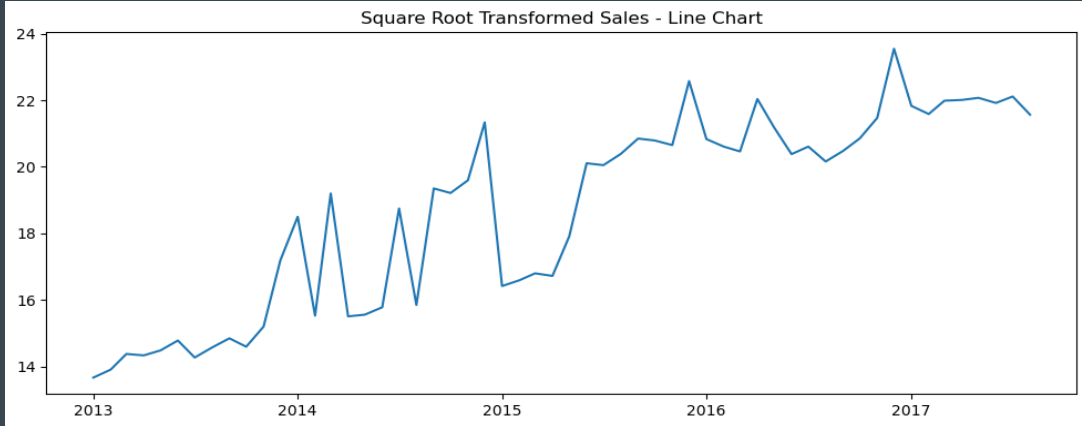

Different_Models
Simple_Exp
Double_Exp
Triple_Exp_Add_Mul
Triple_Exp_Mul
Triple_Exp_Add
Triple_Exp_Mul_Add
0
10
20
30
40
50
Training RMSE

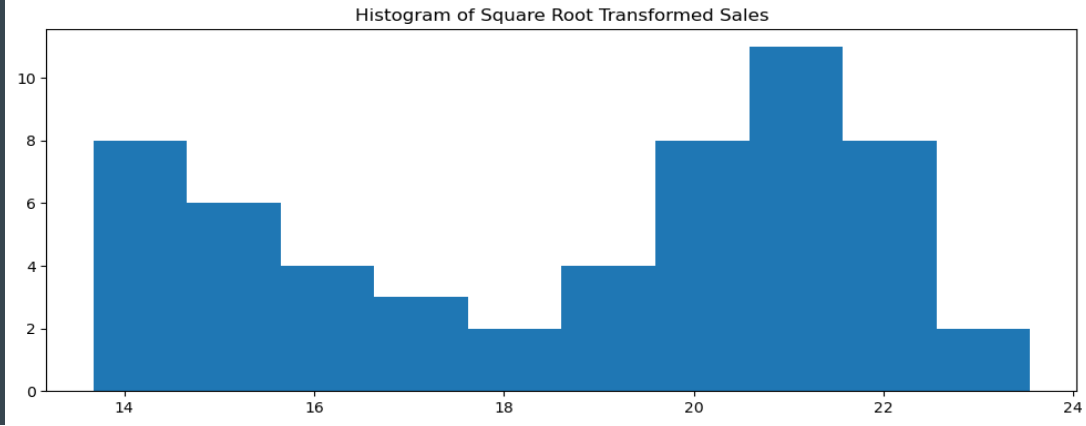

Different_Models
Triple_Exp_Mul
Simple_Exp
Triple_Exp_Add_Mul
Triple_Exp_Mul_Add
Double_Exp
Triple_Exp_Add
0
20
40
60
80
Testing RMSE

Test of Stationarity



Feature Scaling (Transformations)
Square Root Transformation





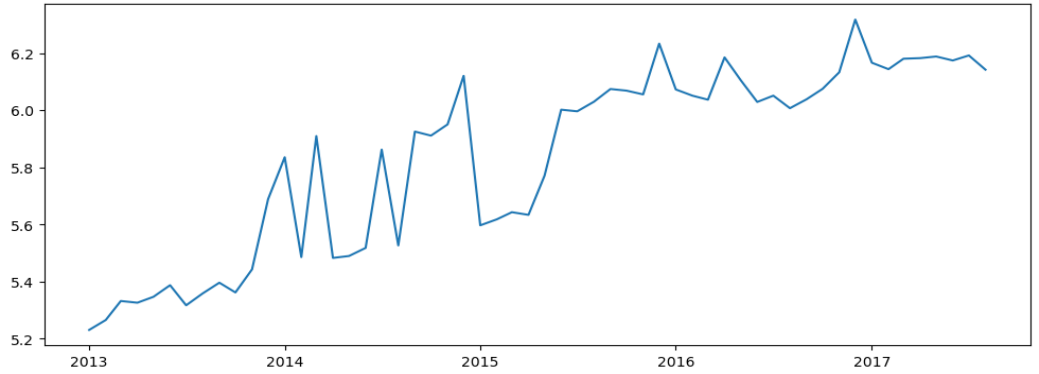

```
In [86]: test_stationarity(square_root['sales'])
```

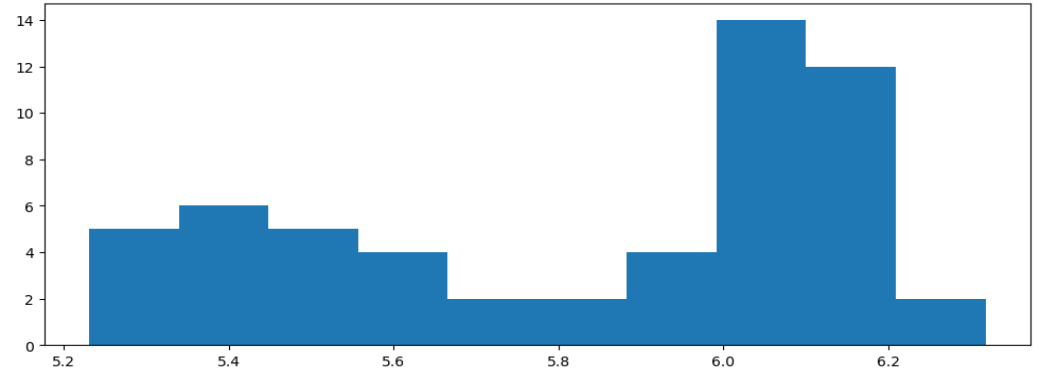
```
Results of Dickey-Fuller Test:
Test Statistic                  -1.647957
p-value                          0.458106
#Lags Used                       1.000000
Number of Observations Used     54.000000
Critical Value (1%)             -3.557709
Critical Value (5%)             -2.916770
Critical Value (10%)            -2.596222
dtype: float64
```

```
In [87]: adf_test(square_root['sales'])
```

```
Fail to reject the null hypothesis
Data is non-stationary
```

Log Transform





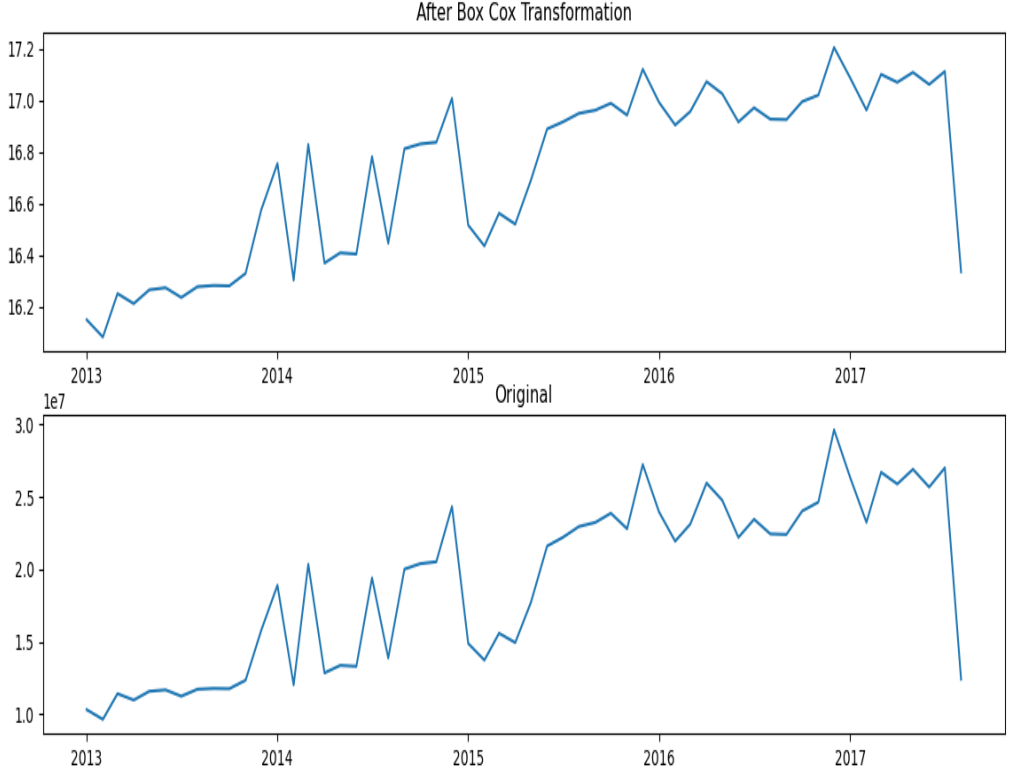

```
In [89]: test_stationarity(log['sales'])
```

```
Results of Dickey-Fuller Test:
Test Statistic                  -1.748199
p-value                          0.406466
#Lags Used                       1.000000
Number of Observations Used     54.000000
Critical Value (1%)             -3.557709
Critical Value (5%)             -2.916770
Critical Value (10%)            -2.596222
dtype: float64
```

```
In [90]: adf_test(log['sales'])
```

```
Fail to reject the null hypothesis
Data is non-stationary
```

Converting Non-Stationary Time Series into Stationary
Box Cox Transformation



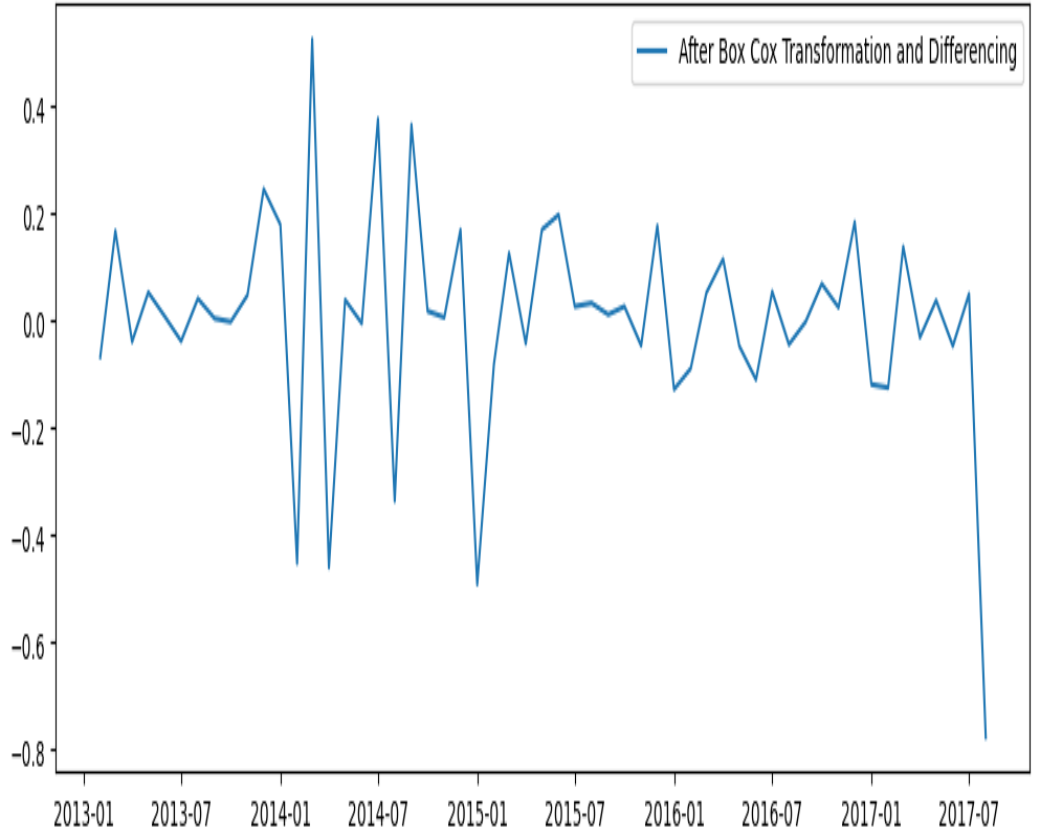

```
n [92]: test_stationarity(data_boxcox)
```

```
Results of Dickey-Fuller Test:
Test Statistic                  -2.185525
p-value                          0.211541
#Lags Used                       1.000000
Number of Observations Used     54.000000
Critical Value (1%)             -3.557709
Critical Value (5%)             -2.916770
Critical Value (10%)            -2.596222
dtype: float64
```

```
n [93]: adf_test(data_boxcox)
```

```
Fail to reject the null hypothesis
Data is non-stationary
```

Differencing of the Box-Cox Transformation



In [95]:
```
#check this log transfered data with function
test_stationarity(data_boxcox_diff)
```

```
Results of Dickey-Fuller Test:
Test Statistic                -1.054889e+01
p-value                        8.274675e-19
#Lags Used                     0.000000e+00
Number of Observations Used    5.400000e+01
Critical Value (1%)           -3.557709e+00
Critical Value (5%)           -2.916770e+00
Critical Value (10%)          -2.596222e+00
dtype: float64
```

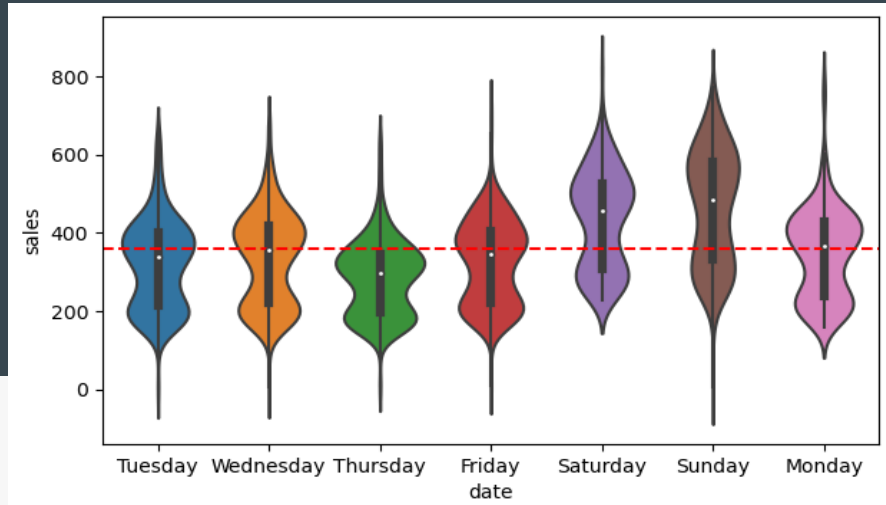
In [96]:
```
adf_test(data_boxcox_diff)
```

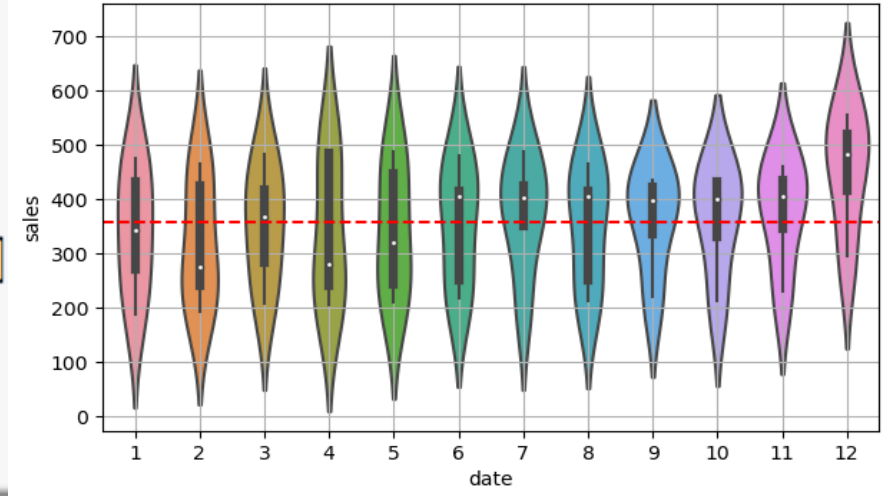
```
Reject the null hypothesis
Data is stationary
```

MODEL BUILDING

Violin plot Of Days Of Week And Month To Determine Variance And Range

```
days_of_week = Daily_sales['sales'].index.day_name()

fig,axes = plt.subplots(2,1,figsize = (7, 9))
sns.violinplot(x=days_of_week, y=Daily_sales['sales'], ax=axes[0])
axes[0].axhline(Daily_sales['sales'].mean(), color='red', linestyle='--')

sns.violinplot(x=monthly_sales.index.month, y=monthly_sales['sales'], ax = axes[1]
axes[1].axhline(monthly_sales['sales'].mean(), color='red', linestyle='--')
plt.grid(True)
plt.show()
```

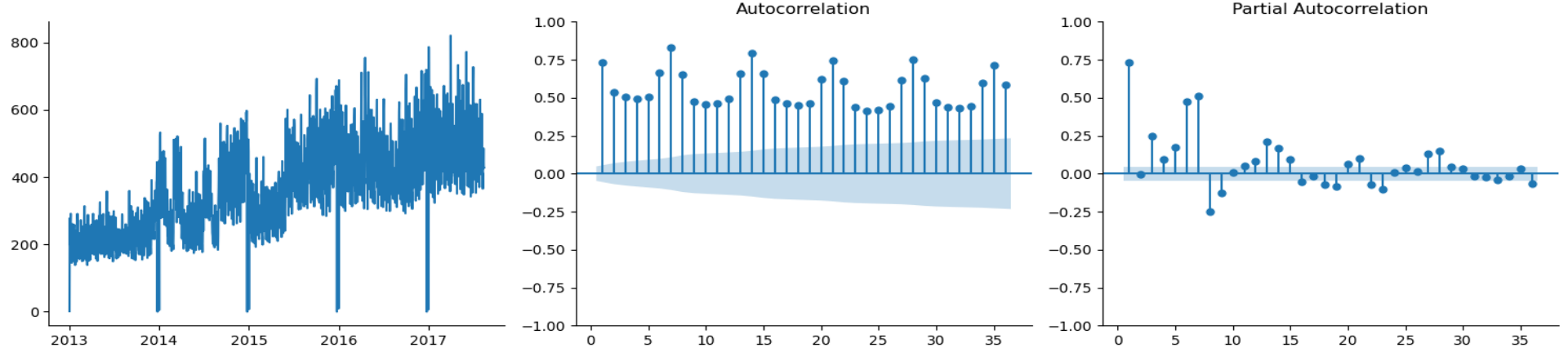




Define Helper Plot Function For Visualization:- Daily Data

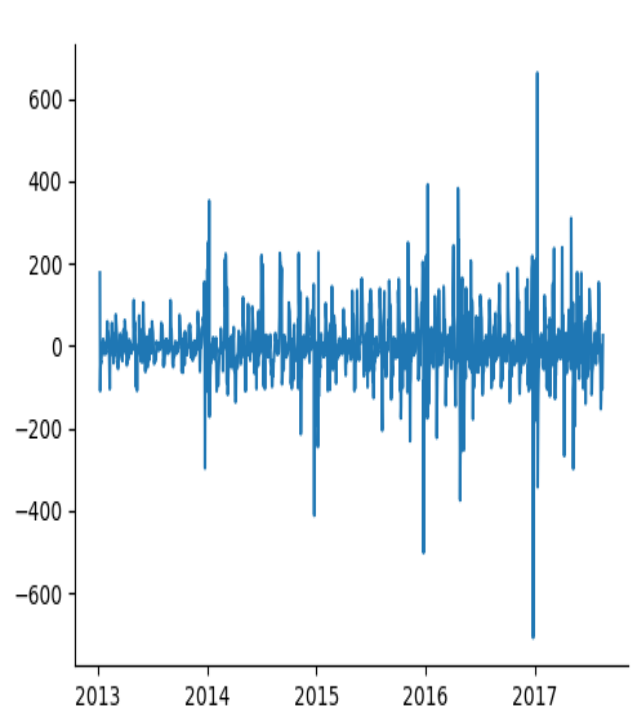
In [162]:
```
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
# define helper plot function for visualization

def plots(data, lags=None):
    plt.figure(figsize=(15, 4))
    layout = (1, 3)
    raw  = plt.subplot2grid(layout, (0, 0))
    acf  = plt.subplot2grid(layout, (0, 1))
    pacf = plt.subplot2grid(layout, (0, 2))

    raw.plot(data)
    plot_acf(data, lags=lags, ax=acf, zero=False)
    plot_pacf(data, lags=lags, ax=pacf, zero = False)
    sns.despine()
    plt.tight_layout()
```

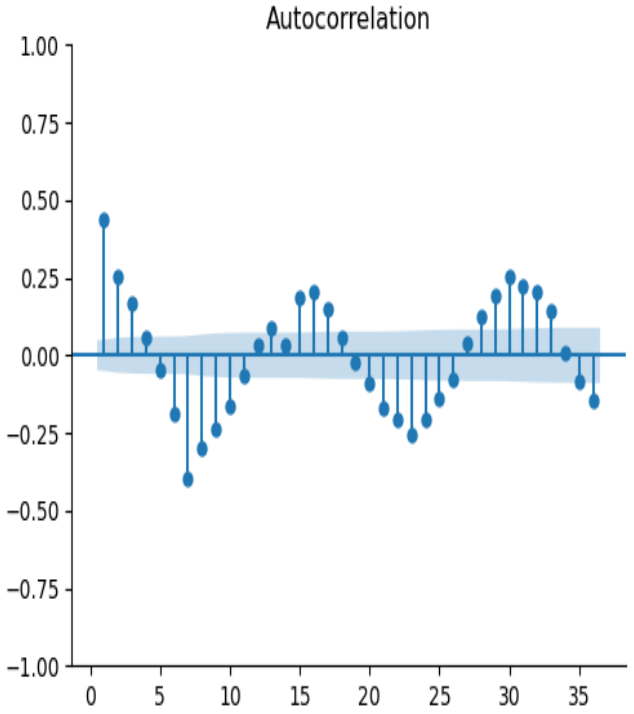
In [166]:
```
plots(daily_sales['sales'], lags = 36)
```

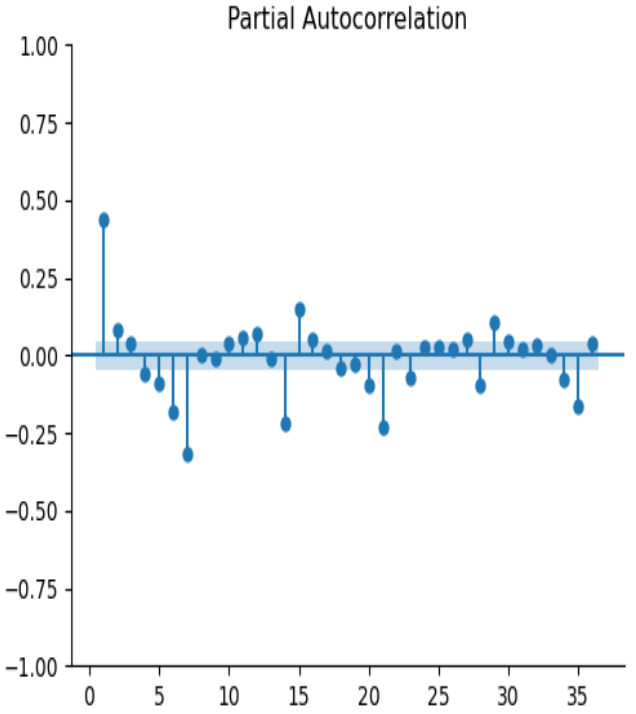


Define Helper Plot Function For Visualization:- Weekly Data

```
daily_sales['weekly_diff'] = daily_sales['sales'].diff(7)
plots(daily_sales.weekly_diff.dropna(), lags=36);`
```



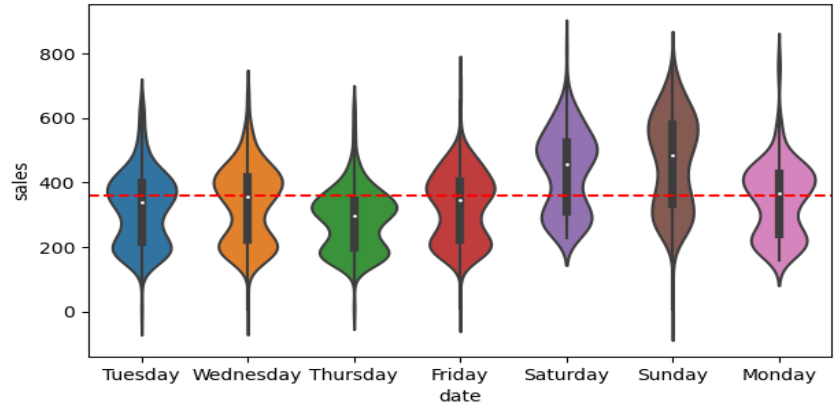


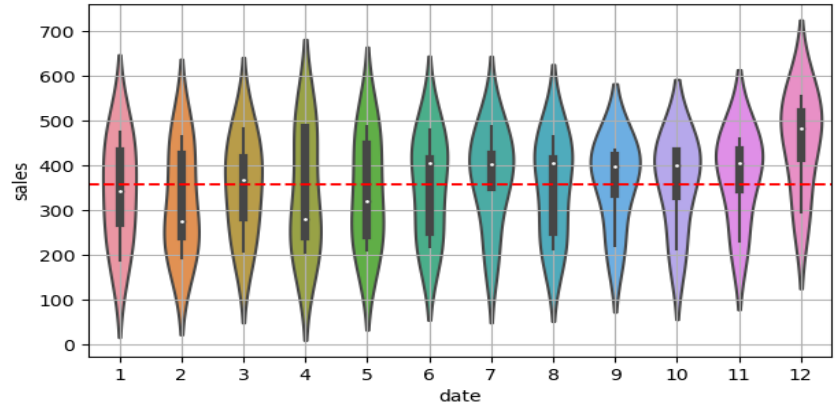


Run SARIMAX

Let's run a grid search to see which parameter combination fits.

Violin plot Of Days Of Week And Month To Determine Variance And Range





SARIMAX Results

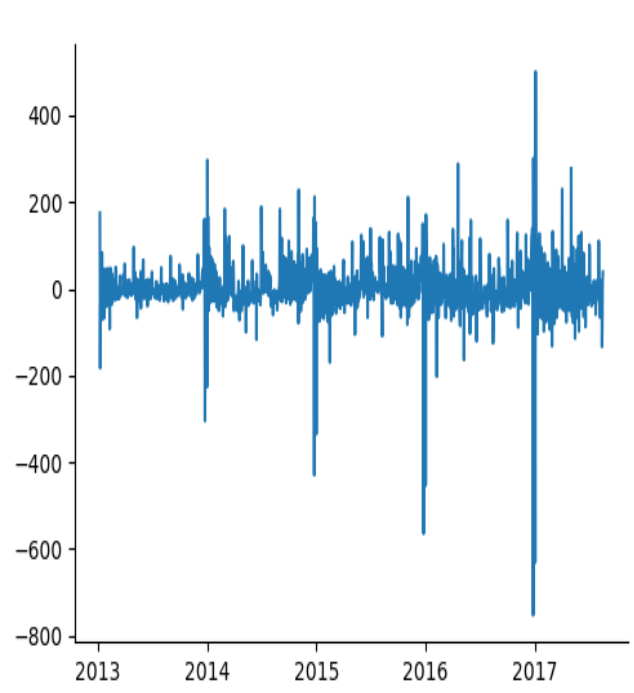
Dep. Variable:	sales	No. Observations:	1688
Model:	SARIMAX(3, 0, 0)x(0, 1, [1], 7)	Log Likelihood	-9233.914
Date:	Fri, 08 Sep 2023	AIC	18479.828
Time:	18:22:15	BIC	18512.390
Sample:	01-01-2013	HQIC	18491.889
	- 08-15-2017		

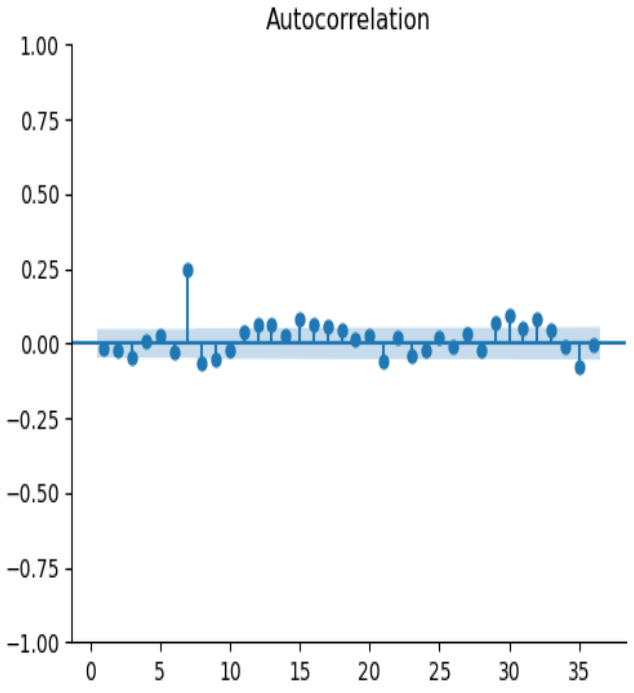
Covariance Type:	opg

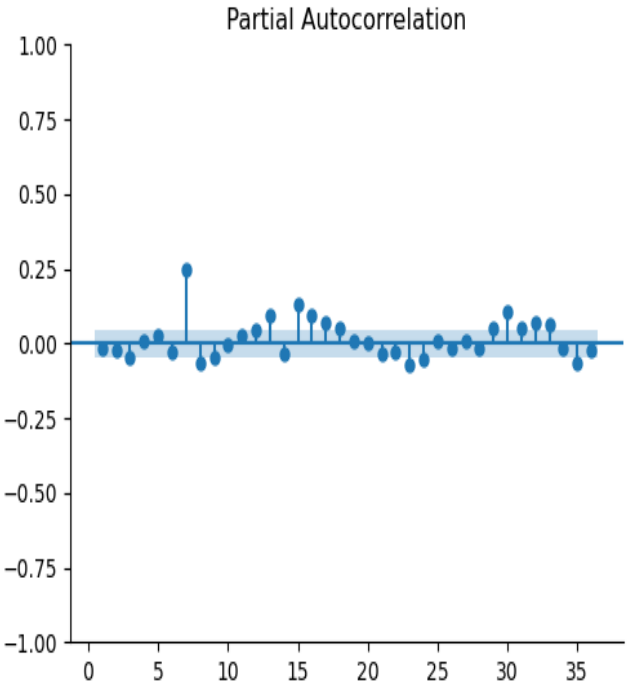
	coef	std err	z	P>|z|	[0.025	0.975]
intercept	0.4550	0.154	2.947	0.003	0.152	0.758
ar.L1	0.3804	0.011	35.791	0.000	0.360	0.401
ar.L2	0.1084	0.018	6.040	0.000	0.073	0.144
ar.L3	0.1532	0.016	9.440	0.000	0.121	0.185
ma.S.L7	-0.9228	0.006	-154.121	0.000	-0.935	-0.911
sigma2	3430.4636	51.580	66.508	0.000	3329.369	3531.558

Ljung-Box (L1) (Q):	0.28	Jarque-Bera (JB):	103225.11
Prob(Q):	0.60	Prob(JB):	0.00
Heteroskedasticity (H):	3.31	Skew:	-2.14

Violin Plot Of Days Of Week And Month To Determine Variance And Range

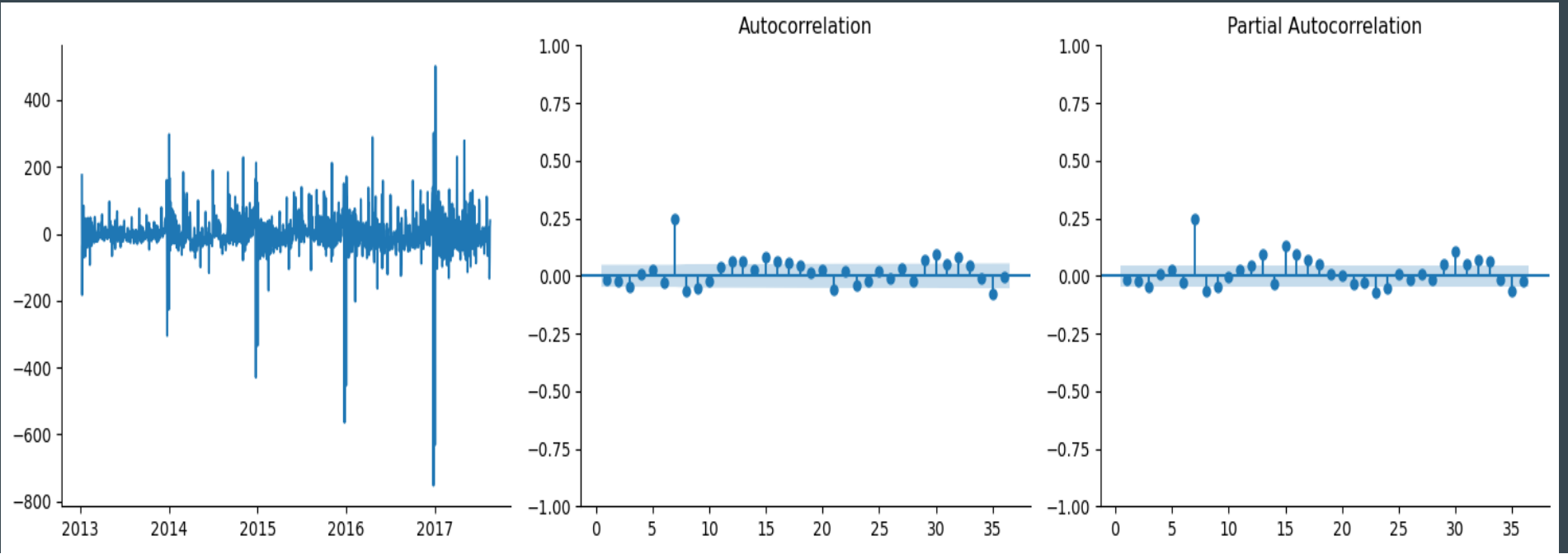






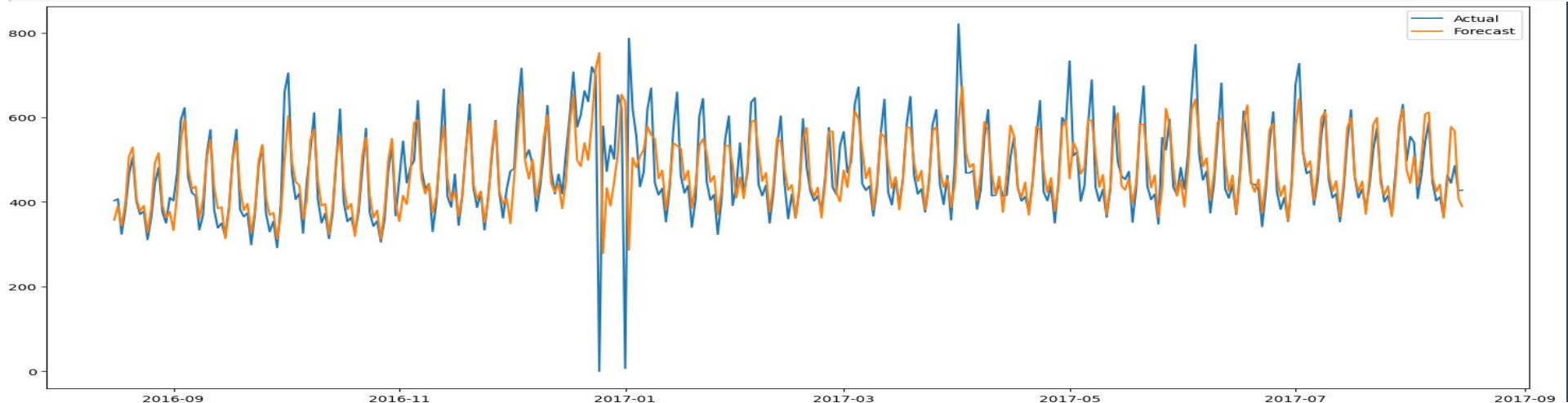
Residual Plot

plots(sar.resid[sar.loglikelihood_burn:], lags=36)

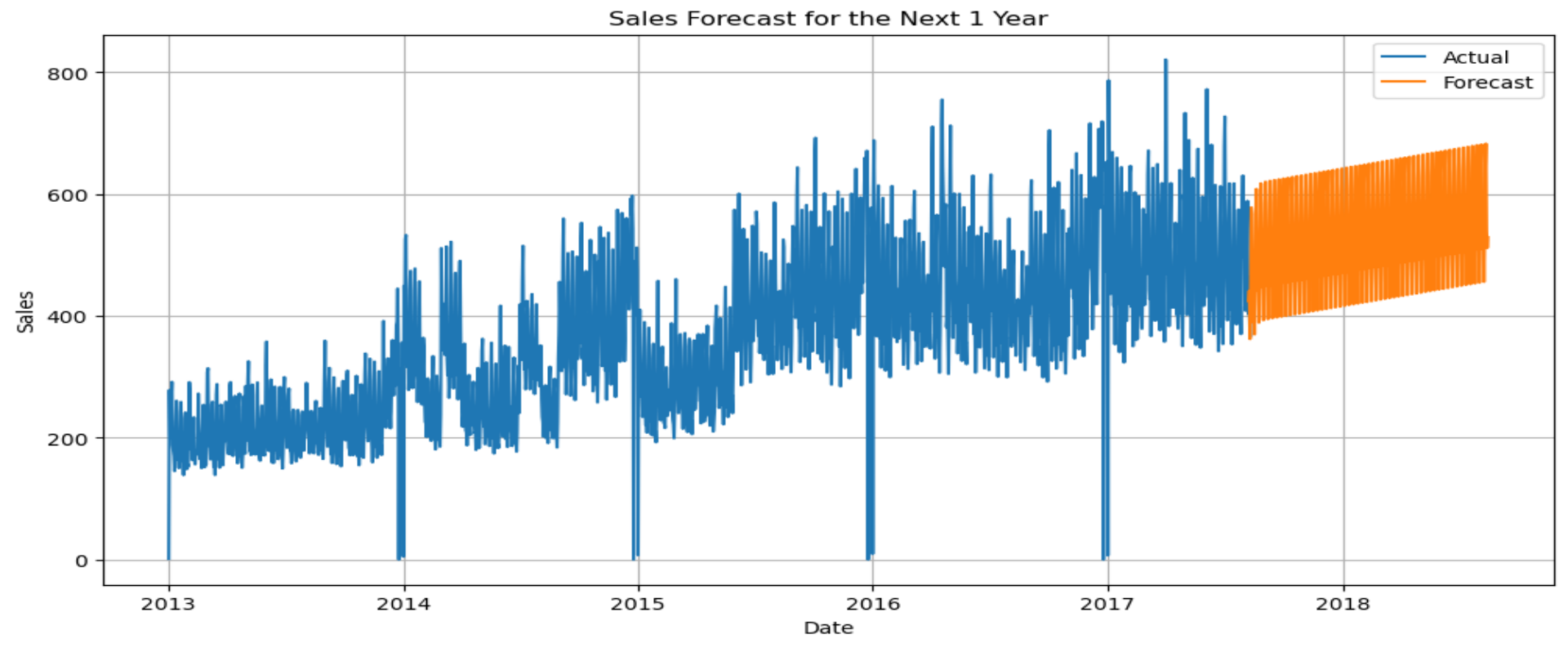


Last 365 Days Prediction

```
# Select the last 365 days of data for comparison
last_365_days = daily_sales.iloc[-365:]

# Get the forecast for the next 60 days
forecast = sar.get_prediction(start=last_365_days.index[0], end=last_365_days.index[-1])

# Extract the predicted values for the next 60 days
predicted_values = forecast.predicted_mean

# Plot the actual sales and predicted values
plt.plot(daily_sales[1323:]['sales'], label = 'Actual')
plt.plot(predicted_values.index, predicted_values, label='Forecast')
plt.legend()
plt.show()
```



Next 365 Days Prediction



Prophet Model

Prophet Model:-**Converting name of culmns**

In [111]:
```
data = pd.DataFrame()
data['y'] = Daily_sales['sales']  # Observed value column
data = data.reset_index()
data = data.rename(columns={'date': 'ds'})
data
```

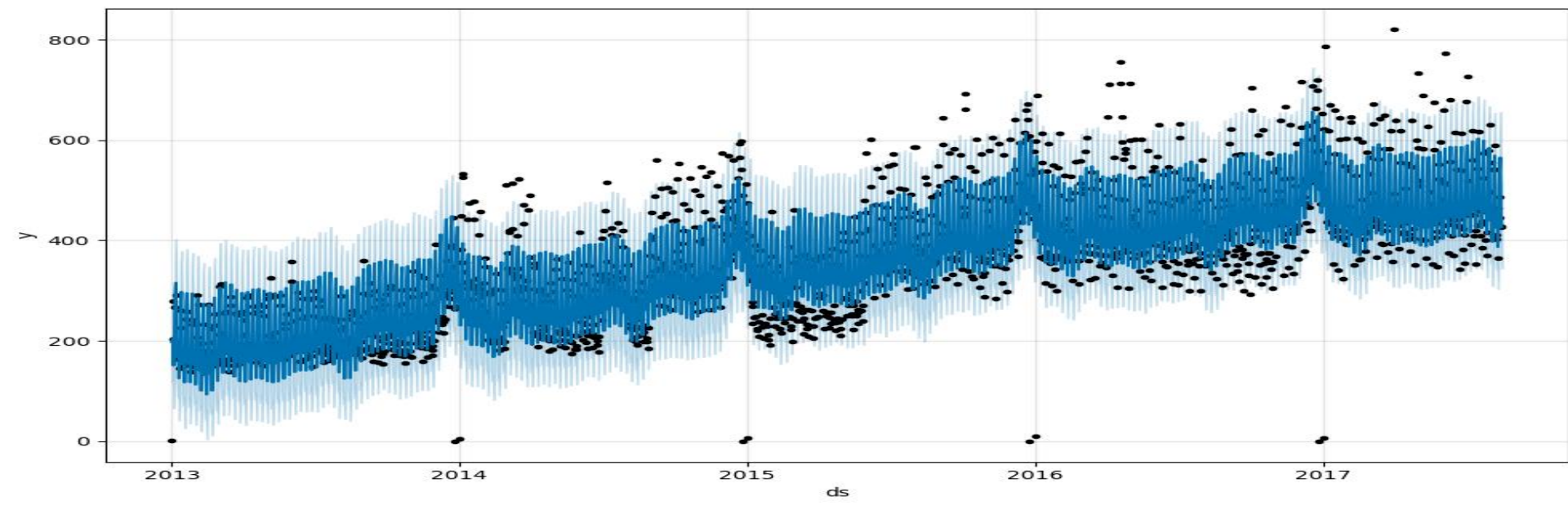
Out[111]:

	ds	y
0	2013-01-01	1.409438
1	2013-01-02	278.390807
2	2013-01-03	202.840197
3	2013-01-04	198.911154
4	2013-01-05	267.873244
...	...	...
1683	2017-08-11	463.733851
1684	2017-08-12	444.798280
1685	2017-08-13	485.768618
1686	2017-08-14	427.004717
1687	2017-08-15	427.980884

1688 rows × 2 columns

Prediction of Till Present Date :- Given Data

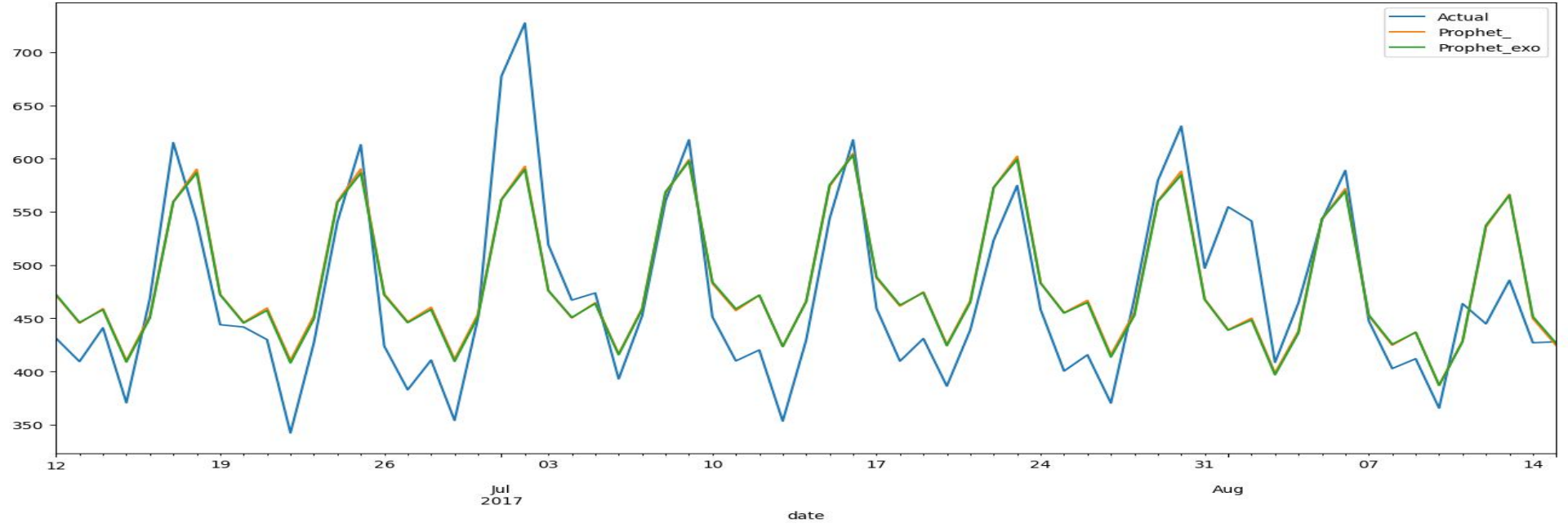
04]:
```
# Make prediction

future1 = m.make_future_dataframe(periods=0)
forecast1 = m.predict(future1)
forecast1[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail()
daily_sales['forecast_2.1'] = forecast1['yhat'].values
```



Last 60 Days Testing:- Training and Testing

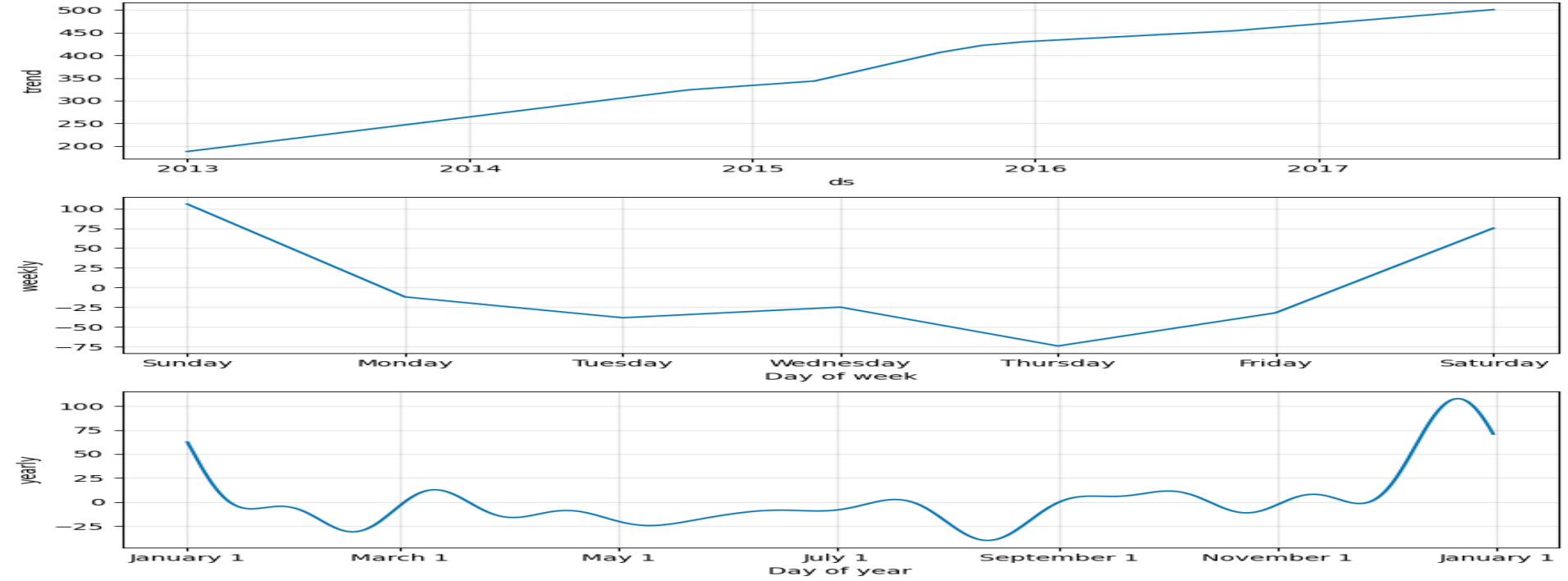
For Better Visualization

```
daily_sales['forecast_2.2'] = forecast['yhat'].values
fig, ax = plt.subplots(figsize=(16, 8))
daily_sales[1623:][['sales', 'forecast_2.1', 'forecast_2.2']].plot(ax=ax);
plt.legend(['Actual', 'Prophet_'])
plt.show()
```



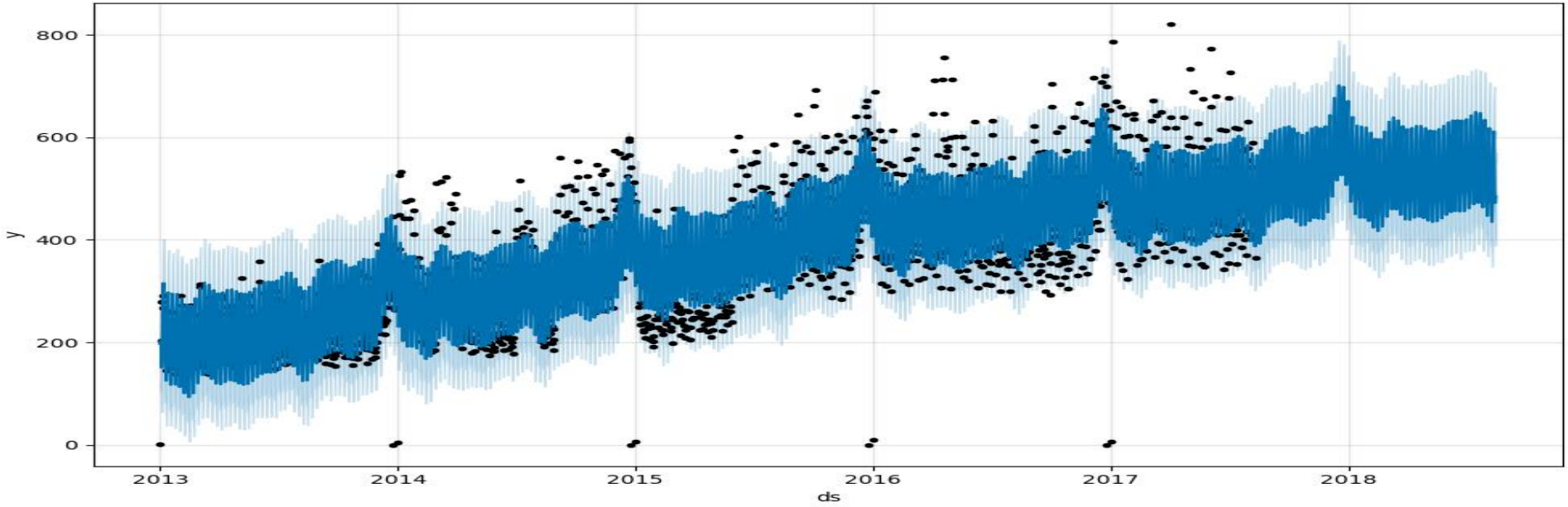
Prediction: -Month, Day of Week, Day of Year

m.plot_components(forecast1)



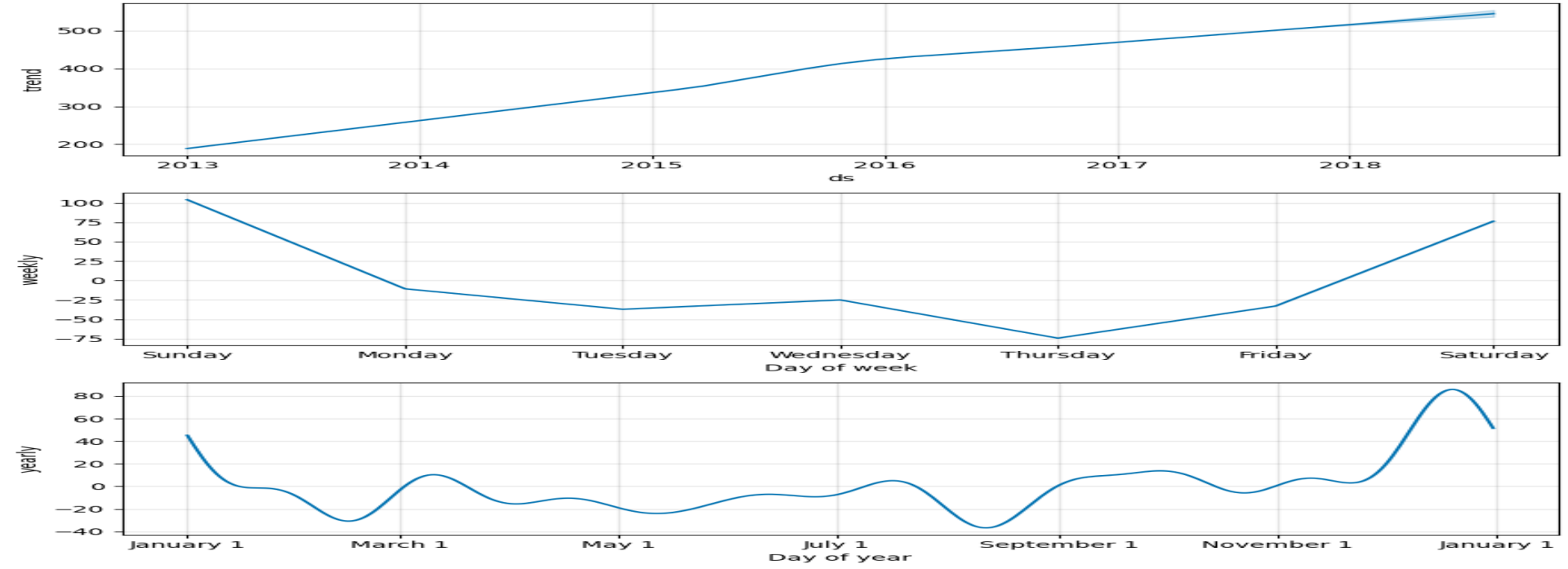
Future Prediction : - 365 Days

```
# Create a dataframe with future dates for forecasting
future = m.make_future_dataframe(periods=365)
# Generate forecasts
forecast = m.predict(future)

# Access the "yhat" column from the forecast
daily_sales['forecast_2.2'] = forecast['yhat']
```



Future Prediction 1 year : - Month Wise, Day Of Week, Day of Year

m.plot_components(forecast)




THANK YOU