Sales Forcasting : Walmart

Table of Contents

---

1. **Problem Statement :**
   A retail store that has multiple outlets across the country are facing issues in managing the inventory - to match the demand with respect to supply. You are a data scientist, who has to come up with useful insights using the data and make prediction models to forecast the sales for X number of months/years.

---

2. **Project Objective :**
   The main objective of the project is to provide useful insights to the retail store for improving inventory management and to develop a sales forecasting model for the next 12 weeks.

---

3. **Dataset Description**

The walmart.csv contains 6435 rows and 8 columns.

Feature Name	Description
Store	Store number
Date	Week of Sales
Weekly_Sales	Sales for the given store in that week
Holiday_Flag	If it is a holiday week
Temperature	Temperature on the day of the sale
Fuel_Price	Cost of the fuel in the region
CPI	Consumer Price Index
Unemployment	Unemployment Rate

---

4. **Data Pre-processing Steps and Inspiration :**

The dataset needs to be pre-processed before it can be used for analysis and modeling. The steps involved are:

- Check for missing values:

> ```
> If any missing values are present, they need to be handled
> ```

- Data cleaning:

> ```
> Check for any anomalies or incosistencies in the data and r
> ```

- Feature engineering:

> ```
> Create new features that may help improve the accuracy of t
> ```

- Data visualization:

> ```
> Visualize the data to gain insights and identify patterns.
> ```

---

5. **Choosing the Algorithm for the Project :**

The algorithm chosen for this project will depend on the type of problem we are trying to solve. For sales forecasting, time series forecasting models such as ARIMA,SARIMA and Prophet can be used.

---

6. **Motivation and Reasons For Choosing the Algorithm :**
   Time series forecasting models are a popular choice for sales forecasting because they take into account the time dimension and the patterns and trends in the data. These models are capable of capturing seasonality,trends and other time-based patterns in the data, which makes them well-suited for this problem.

---

7. **Assumptions :**
   The sales forecasting model assumes that the past patterns and trends in the data will continue into the future and that there are no major changes or disruptions in the market or the company's operatons.

---

8. **Model Evaluation and Techniques :**
   The model will be evaluated using various performance metrics such as Mean Absolute Error, Mean Squared Error(MSE) and Root Mean Error(RMSE). These metrics will help us determine the accuracy of the model and compare it with other models.

---

9. **Inferences from the Same :**
   The insights obtained from the data analysis and modeling can be used to improve inventory management, optimize pricing strategies, and plan promotions and marketing campaigns. The sales forecasting model can help the company make better decisions related to production, supply chain management and budgeting.

---

10. **Future Possibilities of the Project :**

10. **Future Possibilities of the Project :**
The project can be expanded to include more features such as customer demographics, product categories, and competitor data.This will help us build a more comprehensive and accurate sales forecasting model. Additionally, the insights obtained from the analysis can be used to develop personalized marketing and promotional campaigns for specific stores and customer segments.

---

11. **Conlusions:**
In conclusion, sales forecasting is a complex and challenging task, but with the right data and the right algorithms, it can be done effectively. By using the insights obtained from the analysis and the predictions from the sales forecasting model, the company can improve its inventory management and make better decisions related to budgeting and production.
Our analysis shows that sales during holiday weeks are significantly higher than during non-holiday weeks, with sales doubling on average. Additionally, there is a strong seasonal component to the sales data. The average sales of the top performing stores are up to 500% higher than the lowest performing stores.
The best model for predicting future sales is the Random Forest Regressor model,which achieved an RMSE of 1.17e+05. This is a good estimate as it is 88% close to the median sale of the data.
These findings have important implications for businesses as they can inform decisions about inventory, staffing, and marketing efforts. By understanding the factors that drive sales and using a reliable model to forecast future sales, businesses can better plan for the future and optimize their resources.

---

12. **References:**

1. **T**ime Series Forecasting: A Comprehensive Guide - Analytics Vidhya. (2020). Retrieved from https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/
2. Time Series Forecasting: A Practical Guide - DataCamp. (2020). Retrieved from https://www.datacamp.com/community/tutorials/time-series-analysis-tutorial
3. Sales Forecasting: An Overview - Marketing91. (2020). Retrieved from https://www.marketing91.com/sales-forecasting/

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

```
data = pd.read_csv('Walmart.csv')
data.head(2)
```

	Store	Date	Weekly_Sales	Holiday_Flag	Temperature	Fuel_Price	
0	1	05-02-2010	1643690.90	0	42.31	2.572	21
1	1	12-02-2010	1641957.44	1	38.51	2.548	21

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6435 entries, 0 to 6434
```

```
RangeIndex: 6435 entries, 0 to 6434
Data columns (total 8 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Store         6435 non-null   int64  
 1   Date          6435 non-null   object 
 2   Weekly_Sales  6435 non-null   float64
 3   Holiday_Flag  6435 non-null   int64  
 4   Temperature   6435 non-null   float64
 5   Fuel_Price    6435 non-null   float64
 6   CPI           6435 non-null   float64
 7   Unemployment  6435 non-null   float64
dtypes: float64(5), int64(2), object(1)
memory usage: 402.3+ KB
```

```
data['Date'] = pd.to_datetime(data.Date)
```

```
# check duplicates
data[data.duplicated()]
```

Store	Date	Weekly_Sales	Holiday_Flag	Temperature	Fuel_Price	CPI	Unem

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6435 entries, 0 to 6434
Data columns (total 8 columns):
 #   Column        Non-Null Count  Dtype         
---  ------        --------------  -----         
 0   Store         6435 non-null   int64         
 1   Date          6435 non-null   datetime64[ns]
 2   Weekly_Sales  6435 non-null   float64       
 3   Holiday_Flag  6435 non-null   int64         
 4   Temperature   6435 non-null   float64       
 5   Fuel_Price    6435 non-null   float64       
 6   CPI           6435 non-null   float64       
 7   Unemployment  6435 non-null   float64       
dtypes: datetime64[ns](1), float64(5), int64(2)
memory usage: 402.3 KB
```

```
data.columns = [col.lower() for col in data.columns]
col = data.columns
col
```

```
Index(['store', 'date', 'weekly_sales', 'holiday_flag', 'temperature',
       'fuel_price', 'cpi', 'unemployment'],
      dtype='object')
```

```
def find_outlier_rows(data, col, level='both'):
    # compute the interquartile range
    iqr = data[col].quantile(0.75) - data[col].quantile(0.25)
    # compute the upper and lower bounds for identifying outliers
    lower_bound = data[col].quantile(0.25) - 1.5 * iqr
```

```
    upper_bound = data[col].quantile(0.75) + 1.5 * iqr
    # filter the rows based on the level of outliers to return
    if level == 'lower':
        return data[data[col] < lower_bound]
    elif level == 'upper':
        return data[data[col] > upper_bound]
    else:
        return data[(data[col] > upper_bound) | (data[col] < lower_bound)]
```

```
def count_outliers(df):
    # select numeric columns
    df_numeric = df.select_dtypes(include=['int', 'float'])
    # get column names
    columns = df_numeric.columns
    # find the name of all columns with outliers
    outlier_cols = [col for col in columns if len(find_outlier_rows(df_numeric,
    # dataframe to store the results
    outliers_df = pd.DataFrame(columns=['outlier_counts', 'outlier_percent'])
    # count the outliers and compute the percentage of outliers for each column
    for col in outlier_cols:
                outlier_count = len(find_outlier_rows(df_numeric, col))
                all_entries = len(df[col])
                outlier_percent = round(outlier_count * 100 / all_entries, 2
                # store the results in the dataframe
                outliers_df.loc[col] = [outlier_count, outlier_percent]
                # return the resulting dataframe
                return outliers_df
```

count outliers in dataframe using fuctions

```
count_outliers(data).sort_values('outlier_counts',ascending=False)
```

	outlier_counts	outlier_percent
weekly_sales	34.0	0.53

```
find_outlier_rows(data,'weekly_sales').shape
```

```
(34, 8)
```

```
find_outlier_rows(data,'weekly_sales')
```

	store	date	weekly_sales	holiday_flag	temperature	fuel_price
189	2	2010-12-24	3436007.68	0	49.97	2.886
241	2	2011-12-23	3224369.80	0	46.66	3.112
471	4	2010-11-26	2789469.45	1	48.08	2.752

474	4	2010-12-17	2740057.14	0	46.57	2.884
475	4	2010-12-24	3526713.39	0	43.21	2.887
523	4	2011-11-25	3004702.33	1	47.96	3.225
526	4	2011-12-16	2771397.17	0	36.44	3.149
527	4	2011-12-23	3676388.98	0	35.92	3.103
761	6	2010-12-24	2727575.18	0	55.07	2.886
1329	10	2010-11-26	2939946.38	1	55.33	3.162
1332	10	2010-12-17	2811646.85	0	59.15	3.125
1333	10	2010-12-24	3749057.69	0	57.06	3.236
1381	10	2011-11-25	2950198.64	1	60.68	3.760
1385	10	2011-12-23	3487986.89	0	48.36	3.541
1758	13	2010-11-26	2766400.05	1	28.22	2.830
1761	13	2010-12-17	2771646.81	0	35.21	2.842
1762	13	2010-12-24	3595903.20	0	34.90	2.846
1810	13	2011-11-25	2864170.61	1	38.89	3.445
1813	13	2011-12-16	2760346.71	0	27.85	3.282
1814	13	2011-12-23	3556766.03	0	24.76	3.186
1901	14	2010-11-26	2921709.71	1	46.15	3.039
1904	14	2010-12-17	2762861.41	0	30.51	3.140
1905	14	2010-12-24	3818686.45	0	30.59	3.141
1957	14	2011-12-23	3369068.99	0	42.27	3.389
2759	20	2010-11-26	2811634.04	1	46.66	3.039
2761	20	2010-10-12	2752122.08	0	24.27	3.109
2762	20	2010-12-17	2819193.17	0	24.07	3.140
2763	20	2010-12-24	3766687.43	0	25.17	3.141
2811	20	2011-11-25	2906233.25	1	46.38	3.492
2814	20	2011-12-16	2762816.65	0	37.16	3.413
2815	20	2011-12-23	3555371.03	0	40.19	3.389

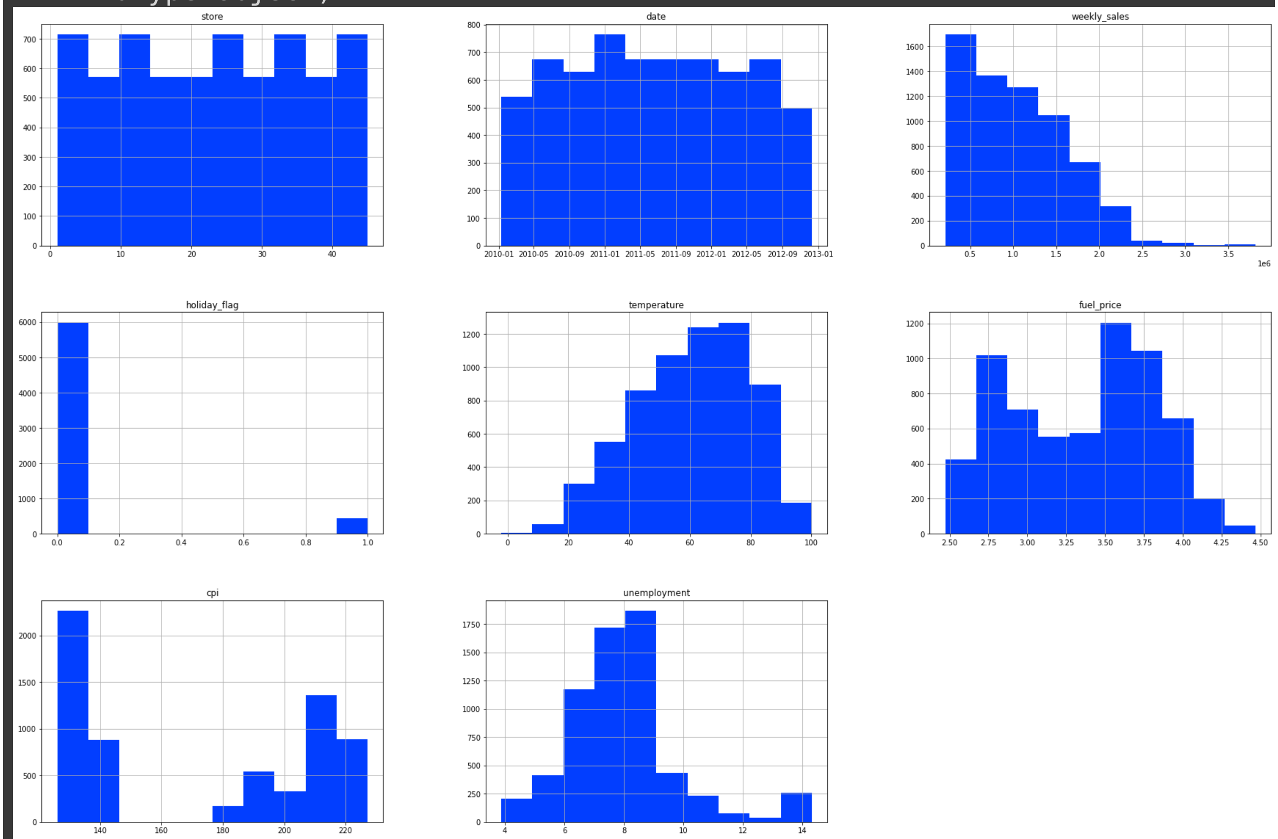
```
data.describe()
```

	store	weekly_sales	holiday_flag	temperature	fuel_price	
count	6435.000000	6.435000e+03	6435.000000	6435.000000	6435.000000	6
mean	23.000000	1.046965e+06	0.069930	60.663782	3.358607	

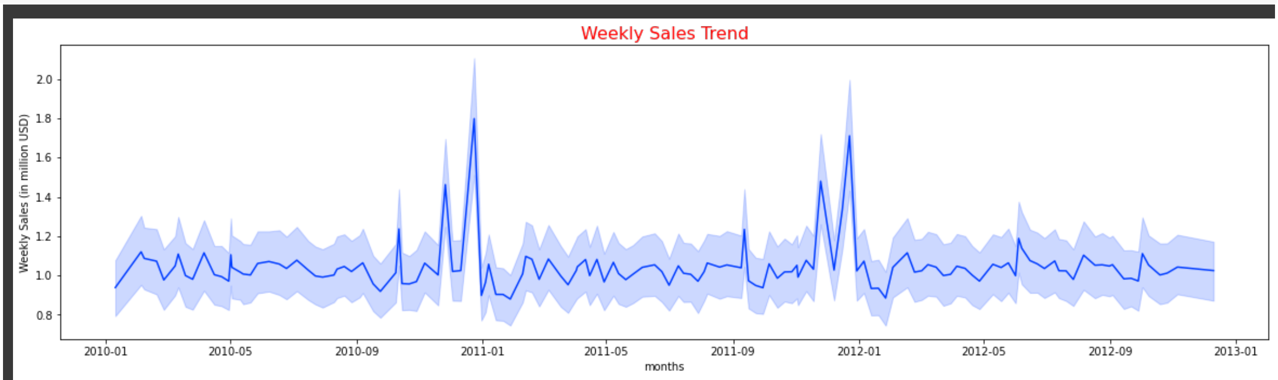
std	12.988182	5.643666e+05	0.255049	18.444933	0.459020
min	1.000000	2.099862e+05	0.000000	-2.060000	2.472000
25%	12.000000	5.533501e+05	0.000000	47.460000	2.933000
50%	23.000000	9.607460e+05	0.000000	62.670000	3.445000
75%	34.000000	1.420159e+06	0.000000	74.940000	3.735000
max	45.000000	3.818686e+06	1.000000	100.140000	4.468000

```
data.hist(figsize=(30,20))
```

```
array([[<AxesSubplot:title={'center':'store'}>,
        <AxesSubplot:title={'center':'date'}>,
        <AxesSubplot:title={'center':'weekly_sales'}>],
       [<AxesSubplot:title={'center':'holiday_flag'}>,
        <AxesSubplot:title={'center':'temperature'}>,
        <AxesSubplot:title={'center':'fuel_price'}>],
       [<AxesSubplot:title={'center':'cpi'}>,
        <AxesSubplot:title={'center':'unemployment'}>, <AxesSubplot:>]],
      dtype=object)
```



```
fig, ax = plt.subplots(figsize=(20, 5))
sns.lineplot(x=data.date, y=(data.weekly_sales/1e6))
plt.xlabel('months')
plt.ylabel('Weekly Sales (in million USD)')
plt.title('Weekly Sales Trend',fontdict={'fontsize': 16, 'color':'red'}, pad=5)
annot = ax.annotate("", xy=(0,0), xytext=(20,20),textcoords="offset points",
                    bbox=dict(boxstyle="round", fc="w"),
                    arrowprops=dict(arrowstyle="->"))
annot.set_visible(False)
plt.show()
```




```
data['employment'] = 100 - data['unemployment']
# split the date column
data['year']= data['date'].dt.year
data['month'] = data['date'].dt.month
data['day'] = data['date'].dt.day
data['day_of_week'] = data['date'].dt.dayofweek
data.head(3)
```

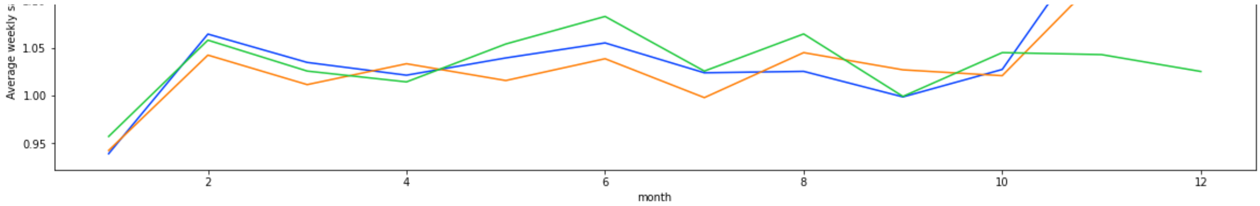
	store	date	weekly_sales	holiday_flag	temperature	fuel_price	
0	1	2010-05-02	1643690.90	0	42.31	2.572	21

1	1	2010-12-02	1641957.44	1	38.51	2.548	2
2	1	2010-02-19	1611968.17	0	39.93	2.514	2

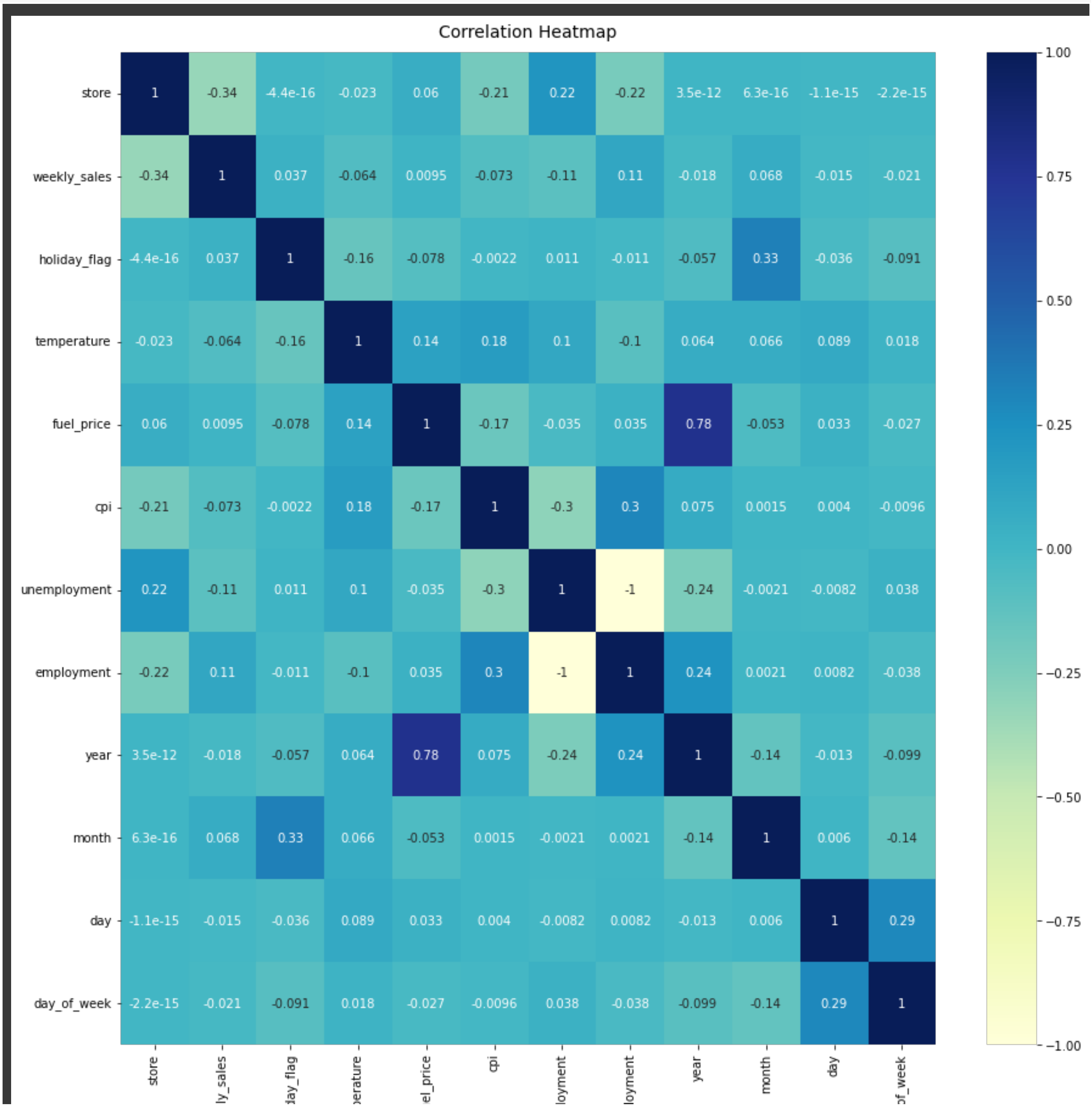
```
# create the pivot table
pivot_table = data.pivot_table(index='month', columns='year', values='weekly_sal
# display the pivot table
pivot_table
```

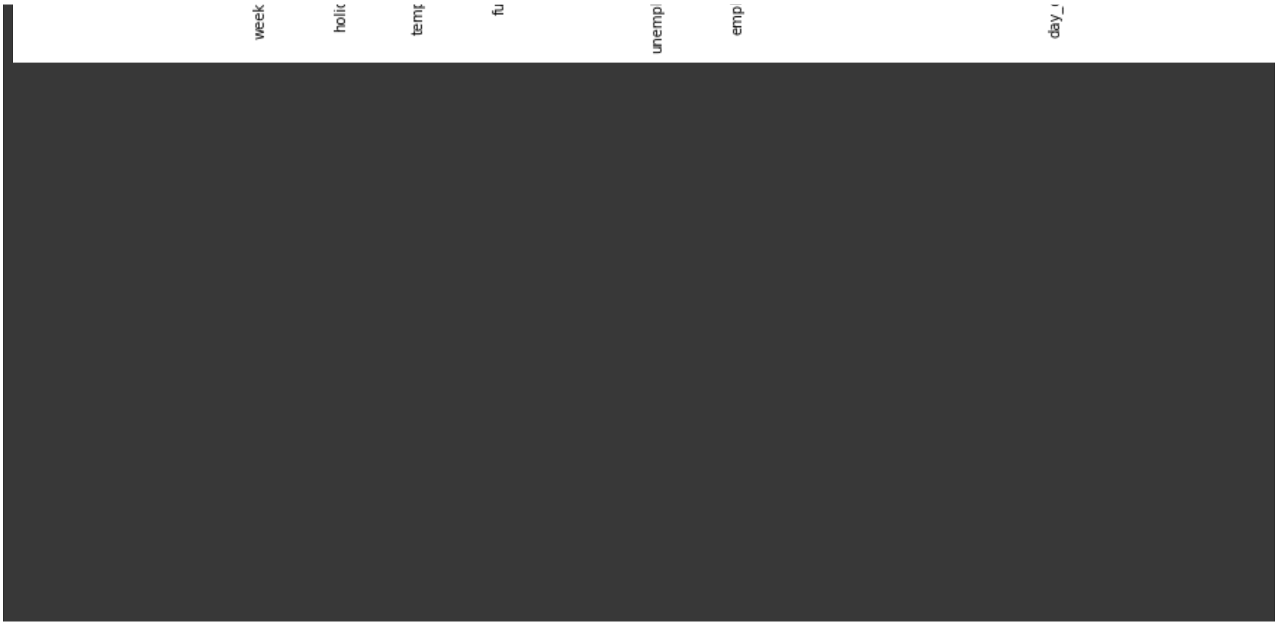
year	**2010**	**2011**	**2012**
month			
1	9.386639e+05	9.420697e+05	9.567817e+05
2	1.064372e+06	1.042273e+06	1.057997e+06
3	1.034590e+06	1.011263e+06	1.025510e+06
4	1.021177e+06	1.033220e+06	1.014127e+06
5	1.039303e+06	1.015565e+06	1.053948e+06
6	1.055082e+06	1.038471e+06	1.082920e+06
7	1.023702e+06	9.976049e+05	1.025480e+06
8	1.025212e+06	1.044895e+06	1.064514e+06
9	9.983559e+05	1.026810e+06	9.988663e+05
10	1.027201e+06	1.020663e+06	1.044885e+06
11	1.176097e+06	1.126535e+06	1.042797e+06
12	1.198413e+06	1.274311e+06	1.025078e+06

```
fig, ax = plt.subplots(figsize=(20, 6))
sns.set_palette("bright")
sns.lineplot(x=pivot_table.index, y=pivot_table[2010]/1e6, ax=ax, label='2010')
sns.lineplot( x=pivot_table.index, y=pivot_table[2011]/1e6, ax=ax, label='2011')
sns.lineplot( x=pivot_table.index, y=pivot_table[2012]/1e6, ax=ax, label='2012')
plt.ylabel('Average weekly sales (in millions USD)')
plt.title('Average Sales Trends for 2010, 2011 & 2012', fontdict ={'fontsize':16
                                                                    'color':'blue
                                                                   }
         )
# Add a legend
plt.legend()
plt.show()
```




```
fig, ax = plt.subplots(figsize=(15,15))
heatmap = sns.heatmap(data.corr(), vmin=-1, vmax=1, annot=True, cmap ="YlGnBu")
heatmap.set_title('Correlation Heatmap', fontdict={'fontsize':14}, pad=12);
```






```
test_data = data.copy()
test_data.head()
```

	store	date	weekly_sales	holiday_flag	temperature	fuel_price	
0	1	2010-05-02	1643690.90	0	42.31	2.572	21
1	1	2010-12-02	1641957.44	1	38.51	2.548	21
2	1	2010-02-19	1611968.17	0	39.93	2.514	21
3	1	2010-02-26	1409727.59	0	46.63	2.561	21
4	1	2010-05-03	1554806.68	0	46.50	2.625	21

```
test_data.pop('date')
```

```
0      2010-05-02
1      2010-12-02
2      2010-02-19
3      2010-02-26
4      2010-05-03
          ...
6430   2012-09-28
6431   2012-05-10
6432   2012-12-10
6433   2012-10-19
6434   2012-10-26
Name: date, Length: 6435, dtype: datetime64[ns]
```

```
x = test_data.drop('weekly_sales',axis=1)
y = test_data['weekly_sales']
```

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(x)
```

```
x.head(3)
```

	store	holiday_flag	temperature	fuel_price	cpi	unemployment	e
0	1	0	42.31	2.572	211.096358	8.106	
1	1	1	38.51	2.548	211.242170	8.106	
2	1	0	39.93	2.514	211.289143	8.106	

```
from sklearn import preprocessing
from sklearn import utils

#convert y values to categorical values
lab = preprocessing.LabelEncoder()
y_transformed = lab.fit_transform(y)
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_transformed, tes
```

```
from sklearn.metrics import mean_squared_error

def evaluate_model(model, X_train, y_train, X_test, y_test):
    # train
    model.fit(X_train, y_train)
    # predict
    y_pred = model.predict(X_test)
    # calculate MSE
    mse = mean_squared_error(y_test, y_pred)
    # calculate RMSE
    rmse = np.sqrt(mse)
    return rmse
```

```
def evaluate_regressors_rmses(regressors, regressor_names, X_train, y_train, X_t
    # evaluate the models and compute their RMSE on the test data
    rmses = [evaluate_model(regressor, X_train, y_train, X_test, y_test) for reg
    # create a dictionary mapping the names of the regressors to their RMSE
    regressor_rmses = dict(zip(regressor_names, rmses))
    # convert the dictionary to a pandas dataframe
    df = pd.DataFrame.from_dict(regressor_rmses, orient='index')
    # reset the index of the dataframe
    df = df.reset_index()
    # rename the columns of the dataframe
    df.columns = ['regressor_name', 'rmse']
    # sort the dataframe by RMSE in ascending order
    return df.sort_values('rmse', ignore_index=True)
```

```
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import Pipeline
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import LogisticRegression

linear_regressor = LinearRegression()
logistic_regression = LogisticRegression()
polynomial_features = PolynomialFeatures(degree=2)
polynomial_regressor = Pipeline([("polynomial_features", polynomial_features),
("linear_regression", linear_regressor)])
ridge_regressor = Ridge()
lasso_regressor = Lasso()
elastic_net_regressor = ElasticNet()
decision_tree_regressor = DecisionTreeRegressor()
random_forest_regressor = RandomForestRegressor()
boosted_tree_regressor = GradientBoostingRegressor()
neural_network_regressor = MLPRegressor()
support_vector_regressor = SVR()
grad_regressor = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1,
knn_regressor = KNeighborsRegressor(n_neighbors=5, weights='uniform')
spline_regressor = make_pipeline(PolynomialFeatures(3), LinearRegression())
```

```
regressors = [linear_regressor, logistic_regression, polynomial_regressor, ridge
decision_tree_regressor, random_forest_regressor, boosted_tree_regressor, neural
support_vector_regressor, knn_regressor, spline_regressor]

regressor_names = ["Linear Regression","Logistic_Regression", "Polynomial Regres
"Elastic Net Regression", "Decision Tree Regression", "Random Forest Regressor",
"Boosted Tree Regression", "Neural Network Regression", "Support Vector Regresso
"K-Nearest Neighbour Regression", "Spline Regression"]
```

```
print('\033[1m Table of regressors and their RMSEs')
evaluate_regressors_rmses(regressors, regressor_names, X_train, y_train, X_test,
```

```
 Table of regressors and their RMSEs
/usr/local/lib/python3.8/dist-packages/sklearn/neural_network/_multilayer_p

Stochastic Optimizer: Maximum iterations (200) reached and the optimization


                  regressor_name         rmse
```

0	Random Forest Regressor	313.787197
1	Decision Tree Regression	389.077048
2	Boosted Tree Regression	558.903563
3	Spline Regression	1468.925226
4	Polynomial Regression	1578.365314
5	K-Nearest Neighbour Regression	1613.757114
6	Ridge Regression	1747.785042
7	Lasso	1747.971167
8	Linear Regression	1748.101011
9	Elastic Net Regression	1772.306088
10	Neural Network Regression	1838.821009
11	Support Vector Regressor	1865.410268
12	Logistic_Regression	2207.653307

```
# !pip install lazypredict
```

```
print(X_test)
print(X_train)
print(y_train)
print(y_test)
```

```
[[-0.38499525 -0.27420425 -0.99079799 ...  1.66322793 -1.44522106
   0.29904117]
 [ 0.07699905 -0.27420425  0.4454804  ... -0.14316392  0.95389383
   0.29904117]
 [-1.61698006 -0.27420425  0.28227927 ...  1.06109731  1.41086809
   0.29904117]
 ...
 [ 1.69397911 -0.27420425 -1.80843024 ... -0.44422923 -1.55946462
   1.70110303]
 [-1.61698006 -0.27420425  0.14130819 ... -1.34742516 -1.33097749
  -0.40198976]
 [-0.4619943  -0.27420425 -1.97651114 ... -0.74529454 -1.44522106
   1.70110303]]
[[-1.15498576 -0.27420425  0.79465491 ...  0.760032    0.15418887
   0.29904117]
 [-1.23198481 -0.27420425 -2.16682209 ...  1.36216262 -1.44522106
  -0.40198976]
 [ 1.46298196 -0.27420425  0.03124231 ...  1.66322793 -0.53127253
   1.0000721 ]
 ...
 [ 1.0779867  -0.27420425  1.41221666 ... -0.44422923 -0.87400323
   1.70110303]
 [ 1.15498576  3.64691651 -0.86880379 ...  1.66322793  1.63935523
   0.29904117]
 [-1.23198481 -0.27420425 -1.81005683 ... -1.34742516  0.382676
   0.29904117]]
```

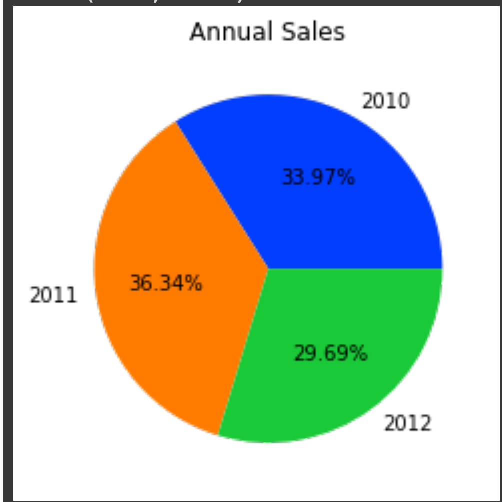
```
  0.29904117]]
[2667 1649 1792 ... 1306  488 1269]
[3853 4374 5553 ... 2870 6146 2611]
```

```
# from lazypredict.Supervised import LazyClassifier
# reg = LazyClassifier(verbose=0, ignore_warnings=True,custom_metric=None)
# models,predictons = reg.fit(X_train,X_test,y_train,y_test)
```

```
# print(models)
```

```
plt.pie(data.groupby('year')['weekly_sales'].sum(),labels=data['year'].unique(),
plt.title('Annual Sales')
```

```
Text(0.5, 1.0, 'Annual Sales')
```




```
import warnings
import itertools
```

```
data
```

	store	date	weekly_sales	holiday_flag	temperature	fuel_price
0	1	2010-05-02	1643690.90	0	42.31	2.572
1	1	2010-12-02	1641957.44	1	38.51	2.548
2	1	2010-02-19	1611968.17	0	39.93	2.514
3	1	2010-02-26	1409727.59	0	46.63	2.561
4	1	2010-05-03	1554806.68	0	46.50	2.625
...	...	...	...	...	...	...
6430	45	2012-09-28	713173.95	0	64.88	3.997
6431	45	2012-05-10	733455.07	0	64.89	3.985
6432	45	2012-12-10	734464.36	0	54.47	4.000
6433	45	2012-10-19	718125.53	0	56.47	3.969
6434	45	2012-10-26	760281.43	0	58.85	3.882

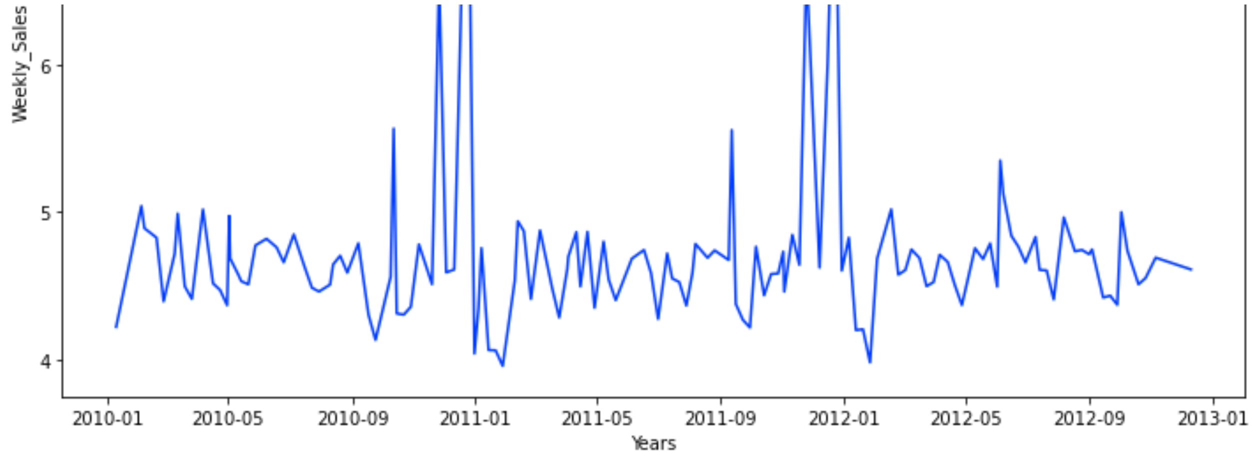
6435 rows × 13 columns

```
week_sales= data.groupby('date')['weekly_sales'].sum()
week_sales
```

```
date
2010-01-10    42239875.87
2010-02-04    50423831.26
2010-02-07    48917484.50
2010-02-19    48276993.78
2010-02-26    43968571.13
                  ...
2012-10-08    47403451.04
2012-10-19    45122410.57
2012-10-26    45544116.29
2012-11-05    46925878.99
2012-12-10    46128514.25
Name: weekly_sales, Length: 143, dtype: float64
```

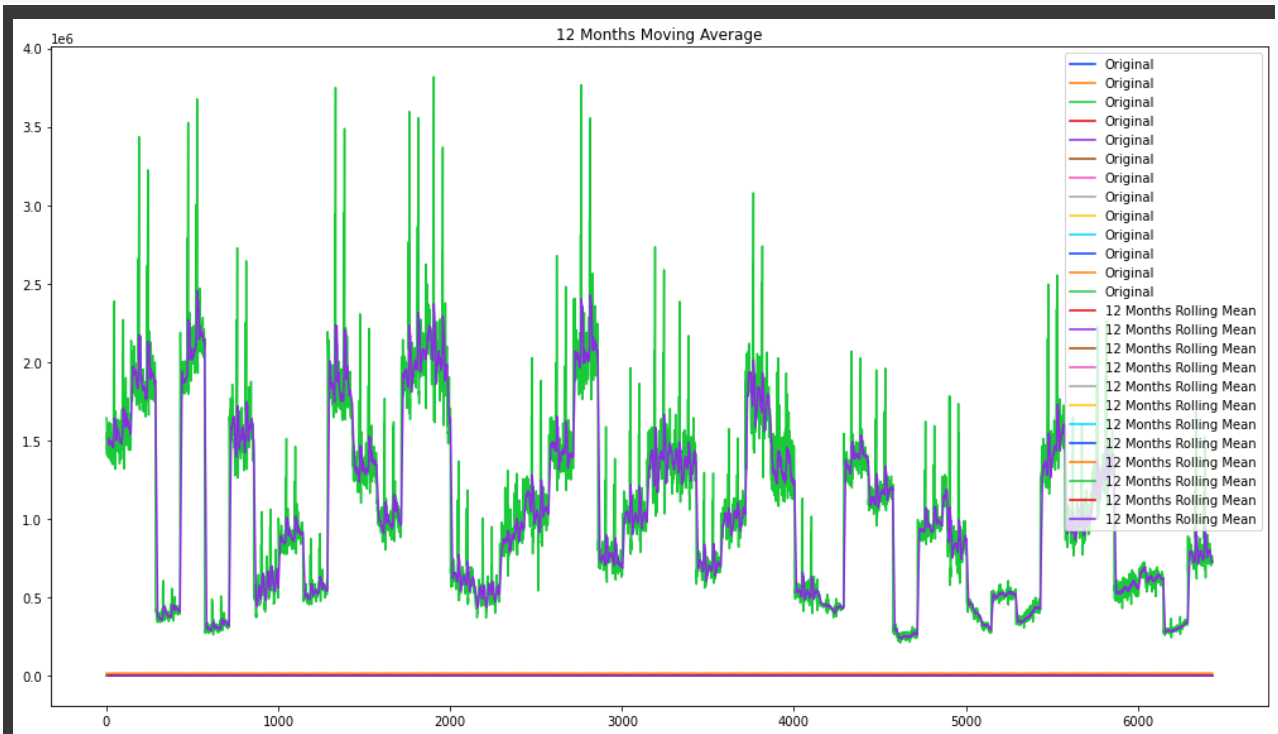
```
plt.figure(figsize=(12,7))
plt.plot(week_sales)
plt.xlabel("Years")
plt.ylabel("Weekly_Sales")
```

```
Text(0, 0.5, 'Weekly_Sales')
```




```
fig,axes = plt.subplots(1,1)
fig.set_figwidth(14)
fig.set_figheight(8)

plt.plot(data.index,data,label='Original')
plt.plot(data.index,data.rolling(window=12).mean(),label='12 Months Rolling Mean
axes.set_title('12 Months Moving Average')
plt.legend(loc='best')
plt.tight_layout()
plt.show()
```



```
rmse = evaluate_regressors_rmses(regressors, regressor_names, X_train, y_train,
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/neural_network/_multilayer_p

Stochastic Optimizer: Maximum iterations (200) reached and the optimization
```

```
# pick the best rmse
best_rmse = rmse.iloc[0]['rmse']
# compute the median of the weekly sales
median_sale = data['weekly_sales'].median()
# compute percentage error
percent_deviation = round((best_rmse*100/median_sale), 2)
# print the result
print('The model has average percentage error of {}%'.format(percent_deviation))
```

```
The model has average percentage error of 0.03%
```

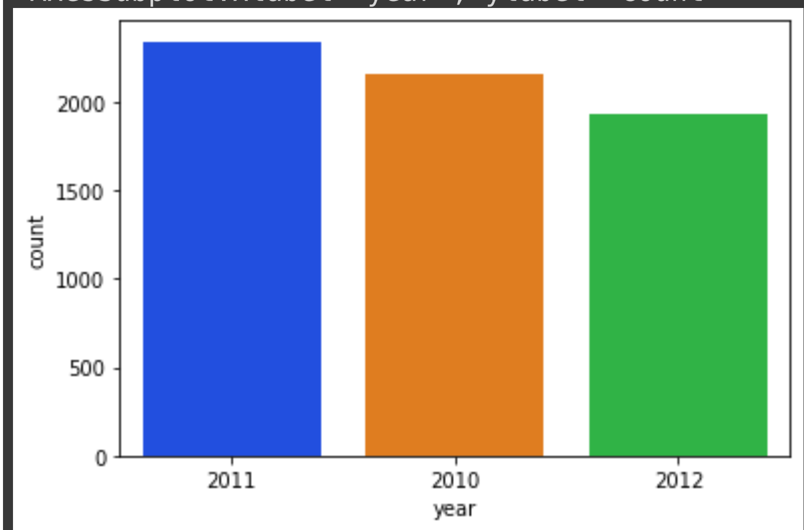
```
# In Random Forest Model for prediction
```

```
sns.countplot(data['year'],order=data['year'].value_counts().index)
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/_decorators.py:36: FutureWar

Pass the following variable as a keyword arg: x. From version 0.12, the onl

<AxesSubplot:xlabel='year', ylabel='count'>
```




```
y_11 = data[data['year']==2011]
y_12 = data[data['year']==2012]
y_10 = data[data['year']==2010]
```

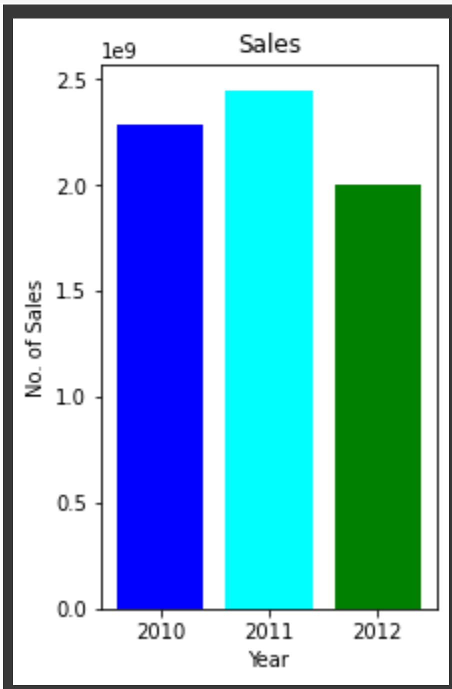
```
a = int(y_10['weekly_sales'].sum())
b = int(y_11['weekly_sales'].sum())
c = int(y_12['weekly_sales'].sum())
```

```
# creating the dataset
total_sales = {2010:a, 2011:b, 2012:c}
year = list(total_sales.keys())
sales = list(total_sales.values())

fig = plt.figure(figsize = (3, 5))

# creating the bar plot
plt.bar(year,sales,color=['blue','cyan','green'])

plt.xlabel("Year")
plt.ylabel("No. of Sales")
plt.title("Sales")
plt.show()
```

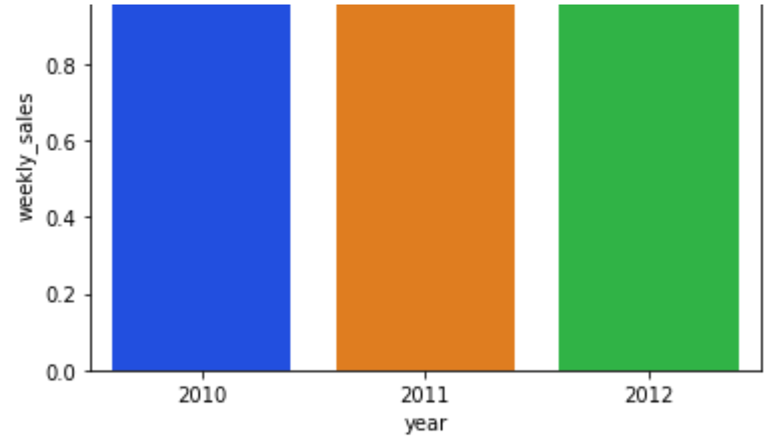



```
# Hear in 2012 the sales has been dropped
```

```
sns.barplot(data["year"], data["weekly_sales"], )
plt.show()
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/_decorators.py:36: FutureWar

Pass the following variables as keyword args: x, y. From version 0.12, the
```




```
data
```

	store	date	weekly_sales	holiday_flag	temperature	fuel_price
0	1	2010-05-02	1643690.90	0	42.31	2.572
1	1	2010-12-02	1641957.44	1	38.51	2.548
2	1	2010-02-19	1611968.17	0	39.93	2.514
3	1	2010-02-26	1409727.59	0	46.63	2.561
4	1	2010-05-03	1554806.68	0	46.50	2.625
...	...	...	...	...	...	...
6430	45	2012-09-28	713173.95	0	64.88	3.997
6431	45	2012-05-10	733455.07	0	64.89	3.985
6432	45	2012-12-10	734464.36	0	54.47	4.000
6433	45	2012-10-19	718125.53	0	56.47	3.969
6434	45	2012-10-26	760281.43	0	58.85	3.882

6435 rows × 13 columns

Timeseries Forecasting using Prophet

```
# from fbprophet import Prophet #you need to install fbprophet using pip install
# fbprophet not able to install
```

```
!pip install prophet
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/cola
Requirement already satisfied: prophet in /usr/local/lib/python3.8/dist-pac
Requirement already satisfied: LunarCalendar>=0.0.9 in /usr/local/lib/pytho
Requirement already satisfied: matplotlib>=2.0.0 in /usr/local/lib/python3.
Requirement already satisfied: convertdate>=2.1.2 in /usr/local/lib/python3
```

```
Requirement already satisfied: pandas>=1.0.4 in /usr/local/lib/python3.8/di
Requirement already satisfied: holidays>=0.14.2 in /usr/local/lib/python3.8
Requirement already satisfied: python-dateutil>=2.8.0 in /usr/local/lib/pyt
Requirement already satisfied: cmdstanpy>=1.0.4 in /usr/local/lib/python3.8
Requirement already satisfied: tqdm>=4.36.1 in /usr/local/lib/python3.8/dis
Requirement already satisfied: numpy>=1.15.4 in /usr/local/lib/python3.8/di
Requirement already satisfied: pymeeus<=1,>=0.3.13 in /usr/local/lib/python
Requirement already satisfied: korean-lunar-calendar in /usr/local/lib/pyth
Requirement already satisfied: hijri-converter in /usr/local/lib/python3.8/
Requirement already satisfied: ephem>=3.7.5.3 in /usr/local/lib/python3.8/d
Requirement already satisfied: pytz in /usr/local/lib/python3.8/dist-packag
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.8/di
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.8/dis
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.
Requirement already satisfied: pyparsing>=2.2.1 in /usr/local/lib/python3.8
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.8/
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.8/dist-pa
WARNING: Running pip as the 'root' user can result in broken permissions an
```

```
from prophet import Prophet
```

```
test_data = data.copy()
test_data=test_data.groupby('date')[['weekly_sales']].sum()
```

```
test_data
```

	weekly_sales
date	
2010-01-10	42239875.87
2010-02-04	50423831.26
2010-02-07	48917484.50
2010-02-19	48276993.78
2010-02-26	43968571.13
...	...
2012-10-08	47403451.04
2012-10-19	45122410.57
2012-10-26	45544116.29
2012-11-05	46925878.99
2012-12-10	46128514.25

143 rows × 1 columns

```
test_data['ds'] = test_data.index
test_data['y'] = test_data['weekly_sales']
```

```
test_data.head()
```

	weekly_sales	ds	y
date			
2010-01-10	42239875.87	2010-01-10	42239875.87
2010-02-04	50423831.26	2010-02-04	50423831.26
2010-02-07	48917484.50	2010-02-07	48917484.50
2010-02-19	48276993.78	2010-02-19	48276993.78
2010-02-26	43968571.13	2010-02-26	43968571.13

```
model = Prophet()
model.fit(test_data)
```

```
INFO:prophet:Disabling daily seasonality. Run prophet with daily_seasonalit
DEBUG:cmdstanpy:input tempfile: /tmp/tmpfwyky4by/3or837tm.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpfwyky4by/6urrb7ty.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.8/dist-packages/prop
08:58:12 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
08:58:12 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
<prophet.forecaster.Prophet at 0x7f3172d16fd0>
```

```
future = model.make_future_dataframe(periods=365)
future.tail()
```

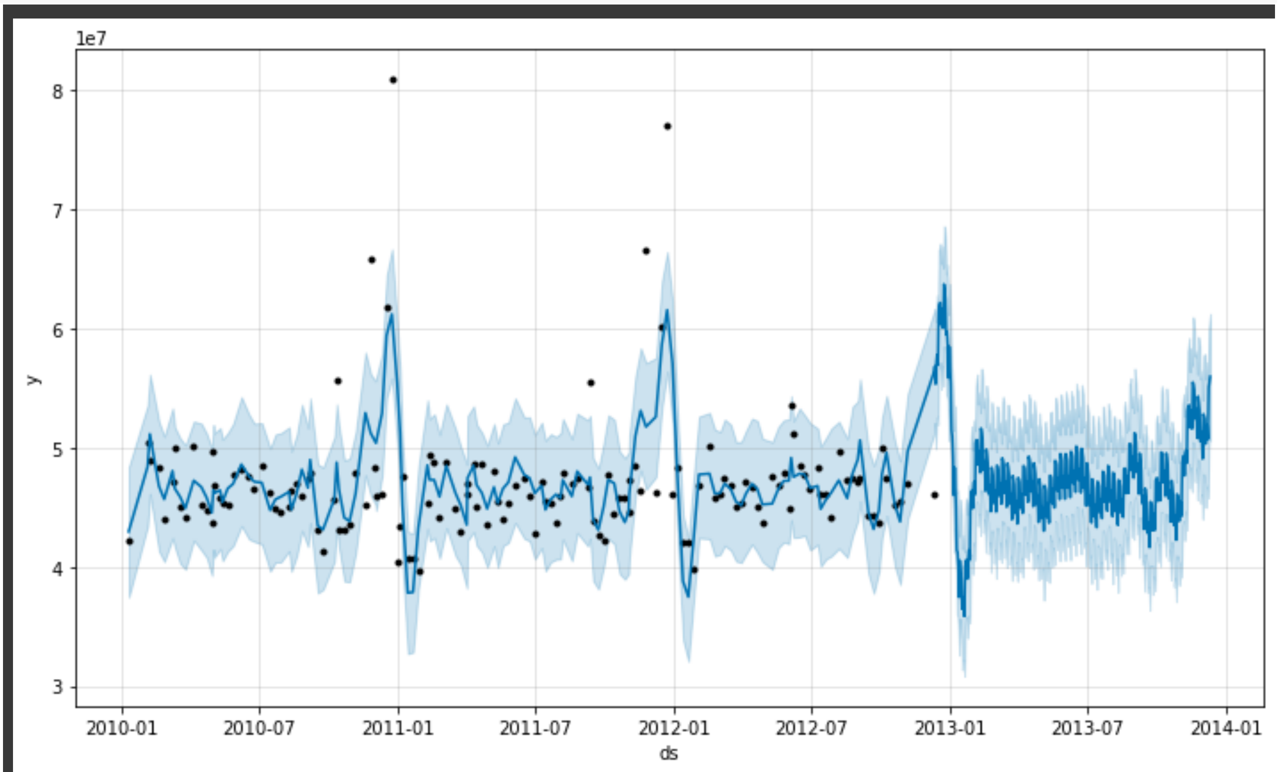
	ds
503	2013-12-06
504	2013-12-07
505	2013-12-08
506	2013-12-09
507	2013-12-10

```
forecast = model.predict(future)
forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail()
```

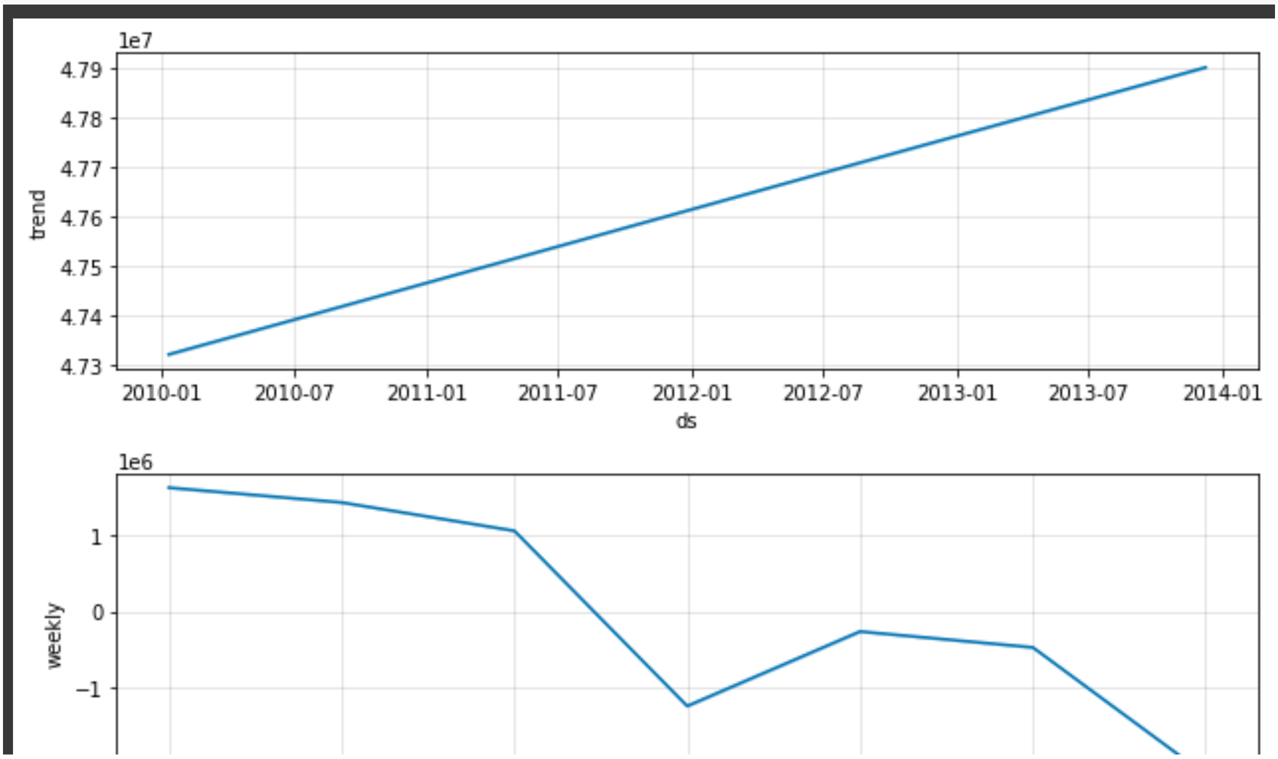
	ds	yhat	yhat_lower	yhat_upper
503	2013-12-06	5.186188e+07	4.675656e+07	5.718532e+07
504	2013-12-07	5.070088e+07	4.580508e+07	5.569579e+07

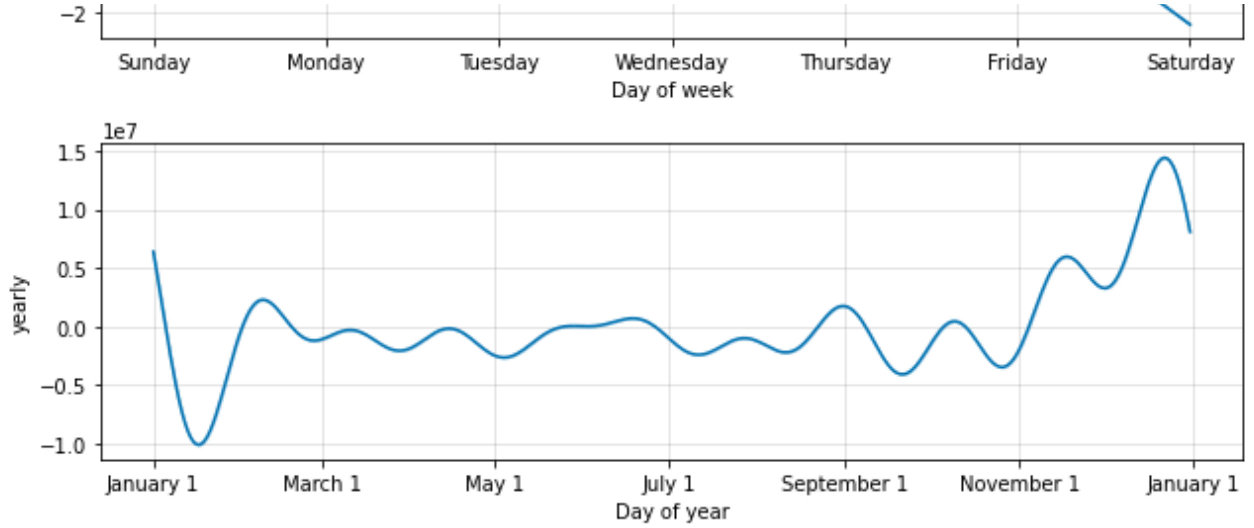
505	2013-12-08	5.510082e+07	4.989662e+07	5.998649e+07
506	2013-12-09	5.559090e+07	5.067787e+07	6.068960e+07
507	2013-12-10	5.596651e+07	5.061194e+07	6.124226e+07

```
fig1 = model.plot(forecast)
```

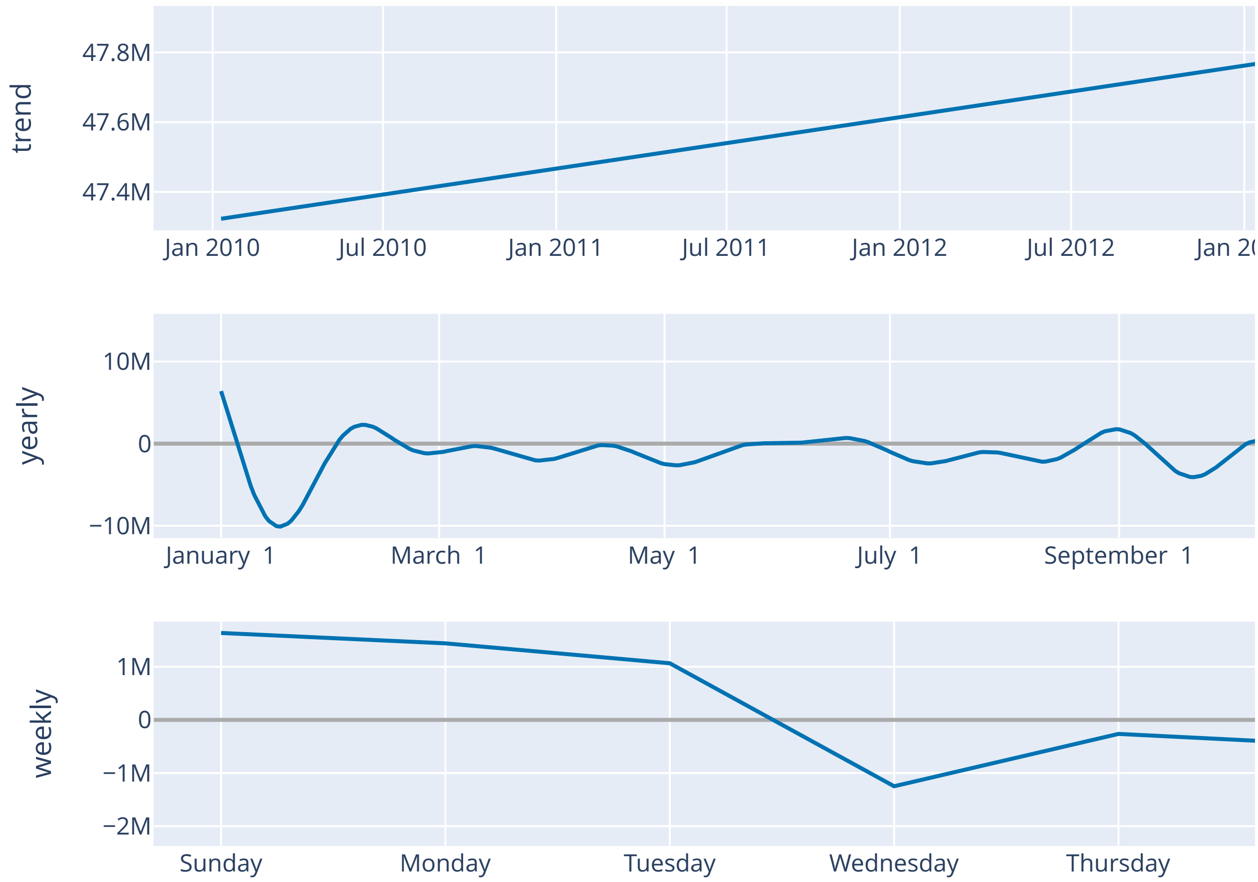



```
fig2 = model.plot_components(forecast)
```






```
from prophet.plot import plot_plotly, plot_components_plotly
plot_plotly(model, forecast)
plot_components_plotly(model, forecast)
```




```
import statsmodels.api as sm
```

```
import statsmodels.tsa.api as smt
import statsmodels.formula.api as smf
```

```
test_data
```

	weekly_sales	ds	y
date			
2010-01-10	42239875.87	2010-01-10	42239875.87
2010-02-04	50423831.26	2010-02-04	50423831.26
2010-02-07	48917484.50	2010-02-07	48917484.50
2010-02-19	48276993.78	2010-02-19	48276993.78
2010-02-26	43968571.13	2010-02-26	43968571.13
...	...	...	...
2012-10-08	47403451.04	2012-10-08	47403451.04
2012-10-19	45122410.57	2012-10-19	45122410.57
2012-10-26	45544116.29	2012-10-26	45544116.29
2012-11-05	46925878.99	2012-11-05	46925878.99
2012-12-10	46128514.25	2012-12-10	46128514.25

143 rows × 3 columns

```
from statsmodels.tsa.stattools import adfuller
adfuller(data)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call
last)
<ipython-input-302-83d5fb234ff8> in <module>
      1 from statsmodels.tsa.stattools import adfuller
----> 2 adfuller(data)

2 frames
/usr/local/lib/python3.8/dist-packages/pandas/core/generic.py in
__array__(self, dtype)
   1991
   1992     def __array__(self, dtype: NpDtype | None = None) ->
np.ndarray:
-> 1993         return np.asarray(self._values, dtype=dtype)
   1994
   1995     def __array_wrap__(

TypeError: float() argument must be a string or a number, not 'Timestamp'
```

```
from statsmodels.tsa.stattools import adfuller
print('Results of Dickey-Fuller Test:')
dftest = adfuller(test_data)
dfoutput = pd.Series(dftest[0:4], index=['Test Statistic', 'p-value', '#lags Use
for key, value in dftest[4].items():
```

```
    dfoutput['Critical Value (%s)'%key] = value
print(dfoutput)
```

```
Results of Dickey-Fuller Test:
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call
last)
<ipython-input-300-edd1235ffb00> in <module>
      1 from statsmodels.tsa.stattools import adfuller
      2 print('Results of Dickey-Fuller Test:')
----> 3 dftest = adfuller(test_data)
      4 dfoutput = pd.Series(dftest[0:4], index=['Test Statistic', 'p-
value', '#lags Used', 'Number of Observations Used'])
      5 for key, value in dftest[4].items():

 2 frames

/usr/local/lib/python3.8/dist-packages/pandas/core/generic.py in
__array__(self, dtype)
   1991
   1992     def __array__(self, dtype: NpDtype | None = None) ->
np.ndarray:
-> 1993         return np.asarray(self._values, dtype=dtype)
   1994
   1995     def __array_wrap__(
```

```
plt.figure(figsize=(18, 6))
plt.plot(sales_log_diff2)
```