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.

In [137…

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

In [138…

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

Out[138]:

	Store	Date	Weekly_Sales	Holiday_Flag	Temperature	Fuel_Price	CPI	Unemployment
0	1	05-02-2010	1643690.90	0	42.31	2.572	211.096358	8.106
1	1	12-02-2010	1641957.44	1	38.51	2.548	211.242170	8.106

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.

In [139…

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

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

In [140…

```
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
```

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 appropriately.`

- Data cleaning:

  > `Check for any anomalies or incosistencies in the data and remove or correct them.`

- Feature engineering:

  > `Create new features that may help improve the accuracy of the sales forecasting model.`

- Data visualization:

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

In [141…

```
data.isnull().sum()
```

Out[141]:

```
Store           0
Date            0
Weekly_Sales    0
Holiday_Flag    0
Temperature     0
Fuel_Price      0
CPI             0
Unemployment    0
dtype: int64
```

In [142…

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

Out[142]:

Store	Date	Weekly_Sales	Holiday_Flag	Temperature	Fuel_Price	CPI	Unemployment

In [143…

```
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
```

In [144…
```
data.columns = [col.lower() for col in data.columns]
col = data.columns
col
```
Out[144]:
```
Index(['store', 'date', 'weekly_sales', 'holiday_flag', 'temperature',
       'fuel_price', 'cpi', 'unemployment'],
      dtype='object')
```
In [145…
```
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)]
```
In [146…
```
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, co
    # 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

In [147…
```
count_outliers(data).sort_values('outlier_counts',ascending=False)
```
Out[147]:

	outlier_counts	outlier_percent
weekly_sales	34.0	0.53

In [148…
```
find_outlier_rows(data,'weekly_sales').shape
```
Out[148]:
```
(34, 8)
```
In [149…
```
find_outlier_rows(data,'weekly_sales')
```

	store	date	weekly_sales	holiday_flag	temperature	fuel_price	cpi	unemploymen
1957	14	2011-12-23	3369068.99	0	42.27	3.389	188.929975	8.52
2759	20	2010-11-26	2811634.04	1	46.66	3.039	204.962100	7.48
2761	20	2010-10-12	2752122.08	0	24.27	3.109	204.687738	7.48
2762	20	2010-12-17	2819193.17	0	24.07	3.140	204.632119	7.48
2763	20	2010-12-24	3766687.43	0	25.17	3.141	204.637673	7.48
2811	20	2011-11-25	2906233.25	1	46.38	3.492	211.412076	7.08
2814	20	2011-12-16	2762816.65	0	37.16	3.413	212.068504	7.08
2815	20	2011-12-23	3555371.03	0	40.19	3.389	212.236040	7.08
3192	23	2010-12-24	2734277.10	0	22.96	3.150	132.747742	5.28
3764	27	2010-12-24	3078162.08	0	31.34	3.309	136.597273	8.02
3816	27	2011-12-23	2739019.75	0	41.59	3.587	140.528765	7.90

In [150…

```
data.describe()
```

Out[150]:

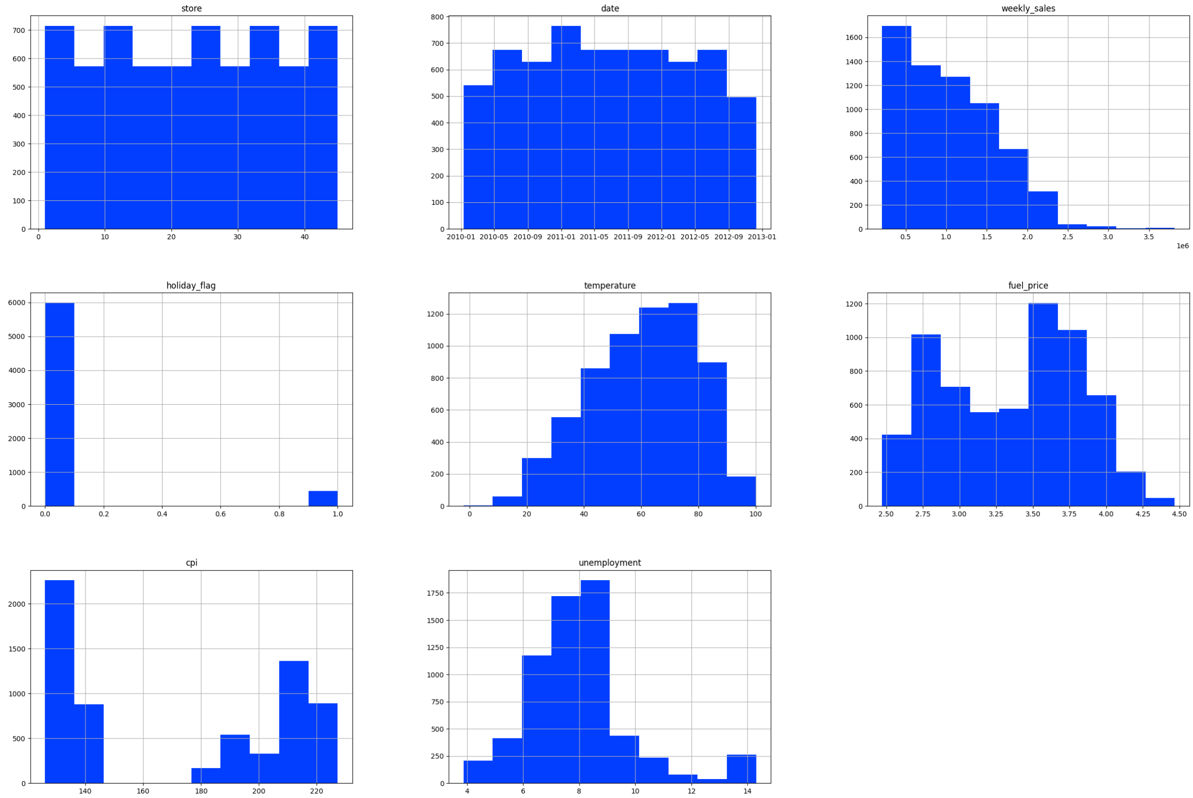
	store	weekly_sales	holiday_flag	temperature	fuel_price	cpi	unemploy
count	6435.000000	6.435000e+03	6435.000000	6435.000000	6435.000000	6435.000000	6435.0
mean	23.000000	1.046965e+06	0.069930	60.663782	3.358607	171.578394	7.9
std	12.988182	5.643666e+05	0.255049	18.444933	0.459020	39.356712	1.8
min	1.000000	2.099862e+05	0.000000	-2.060000	2.472000	126.064000	3.8
25%	12.000000	5.533501e+05	0.000000	47.460000	2.933000	131.735000	6.8
50%	23.000000	9.607460e+05	0.000000	62.670000	3.445000	182.616521	7.8
75%	34.000000	1.420159e+06	0.000000	74.940000	3.735000	212.743293	8.6
max	45.000000	3.818686e+06	1.000000	100.140000	4.468000	227.232807	14.3

In [151…

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

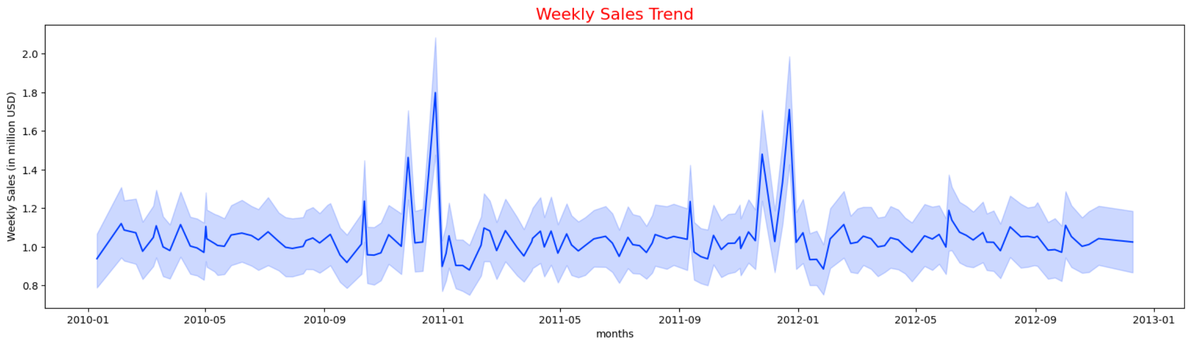
Out[151]:

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



In [152…

```python
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()
```



In [153…

```python
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)
```

Out[153]:

	store	date	weekly_sales	holiday_flag	temperature	fuel_price	cpi	unemployment
0	1	2010-05-02	1643690.90	0	42.31	2.572	211.096358	8.106
1	1	2010-12-02	1641957.44	1	38.51	2.548	211.242170	8.106
2	1	2010-02-19	1611968.17	0	39.93	2.514	211.289143	8.106

In [154…

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

Out[154]:

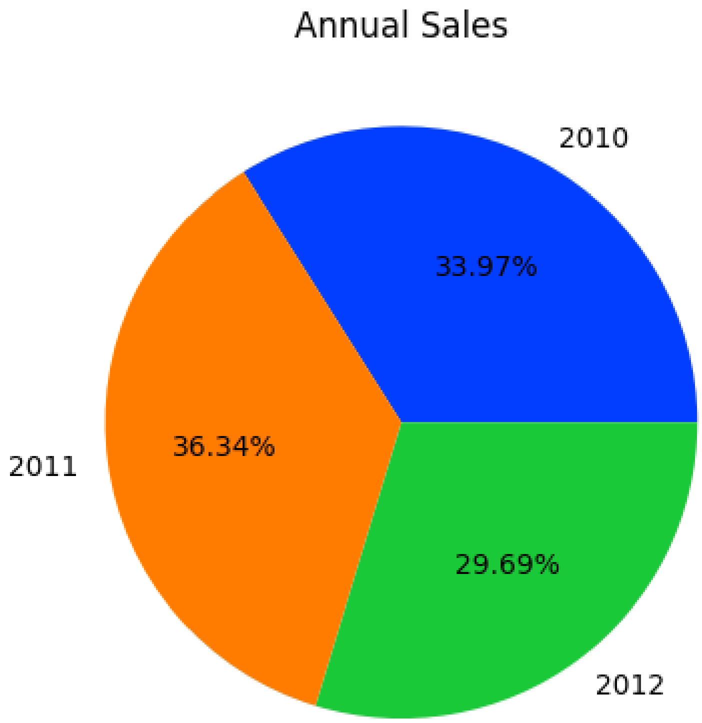
year month	2010	2011	2012
1	9.386639e+05	9.420697e+05	9.567817e+05
2	1.064372e+06	1.042273e+06	1.057997e+06
3	1.034590e+06	1.011263e+06	1.025510e+06
4	1.021177e+06	1.033220e+06	1.014127e+06
5	1.039303e+06	1.015565e+06	1.053948e+06
6	1.055082e+06	1.038471e+06	1.082920e+06
7	1.023702e+06	9.976049e+05	1.025480e+06
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

In [155…

```
plt.pie(data.groupby('year')['weekly_sales'].sum(),labels=data['year'].unique(),no
plt.title('Annual Sales')
```

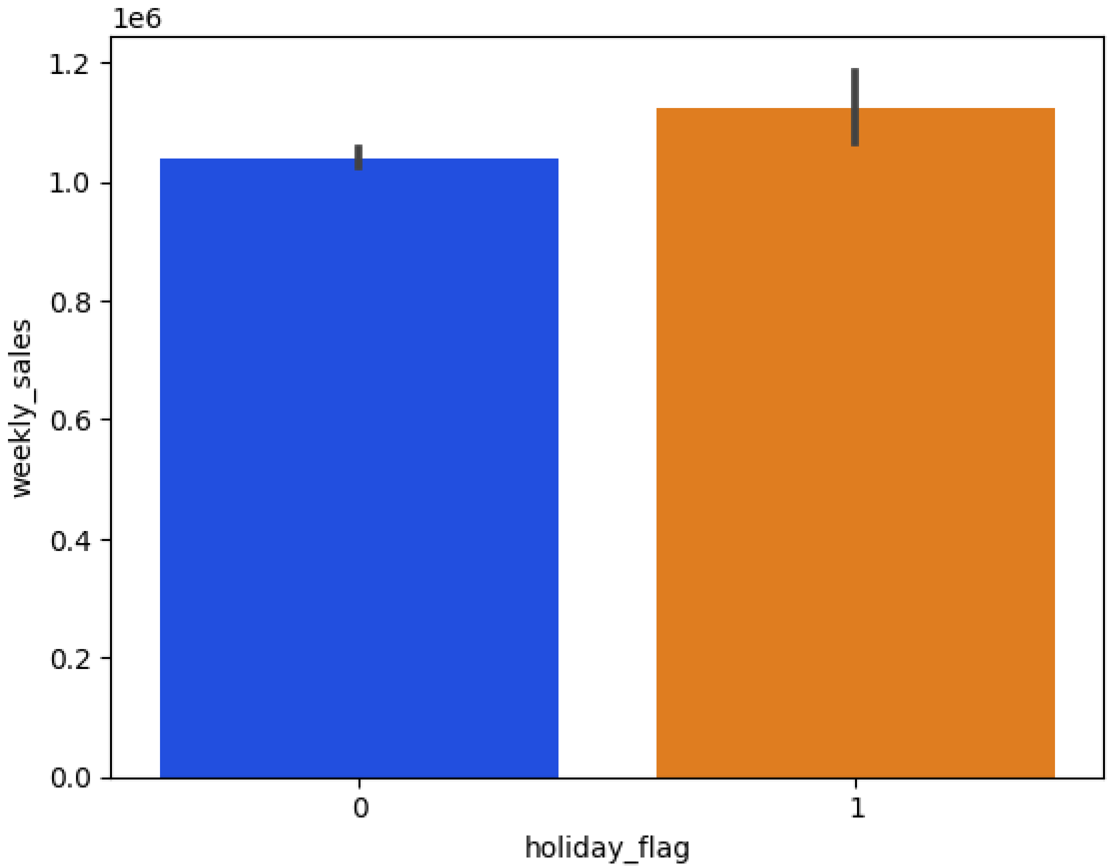
Out[155]:

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



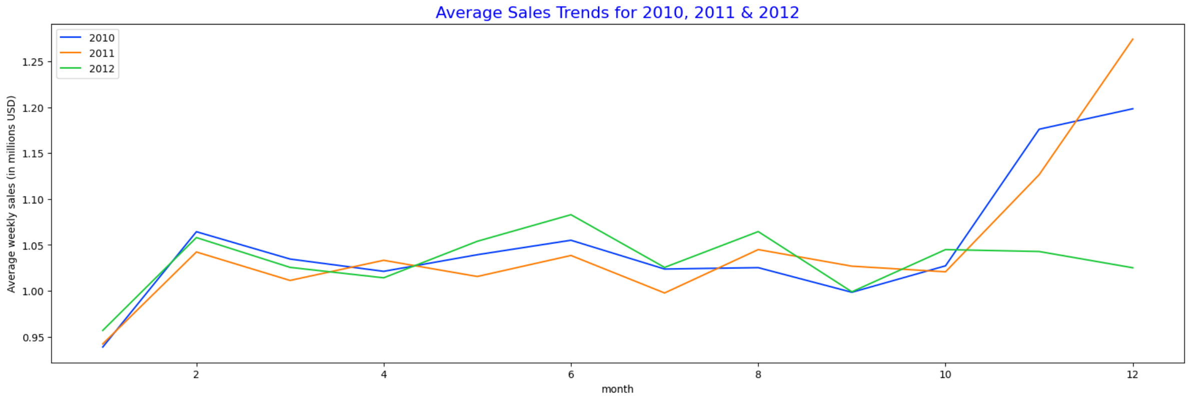
In [156…
```
sns.barplot(x='holiday_flag', y='weekly_sales', data=data)
```

Out[156]:
```
<Axes: xlabel='holiday_flag', ylabel='weekly_sales'>
```



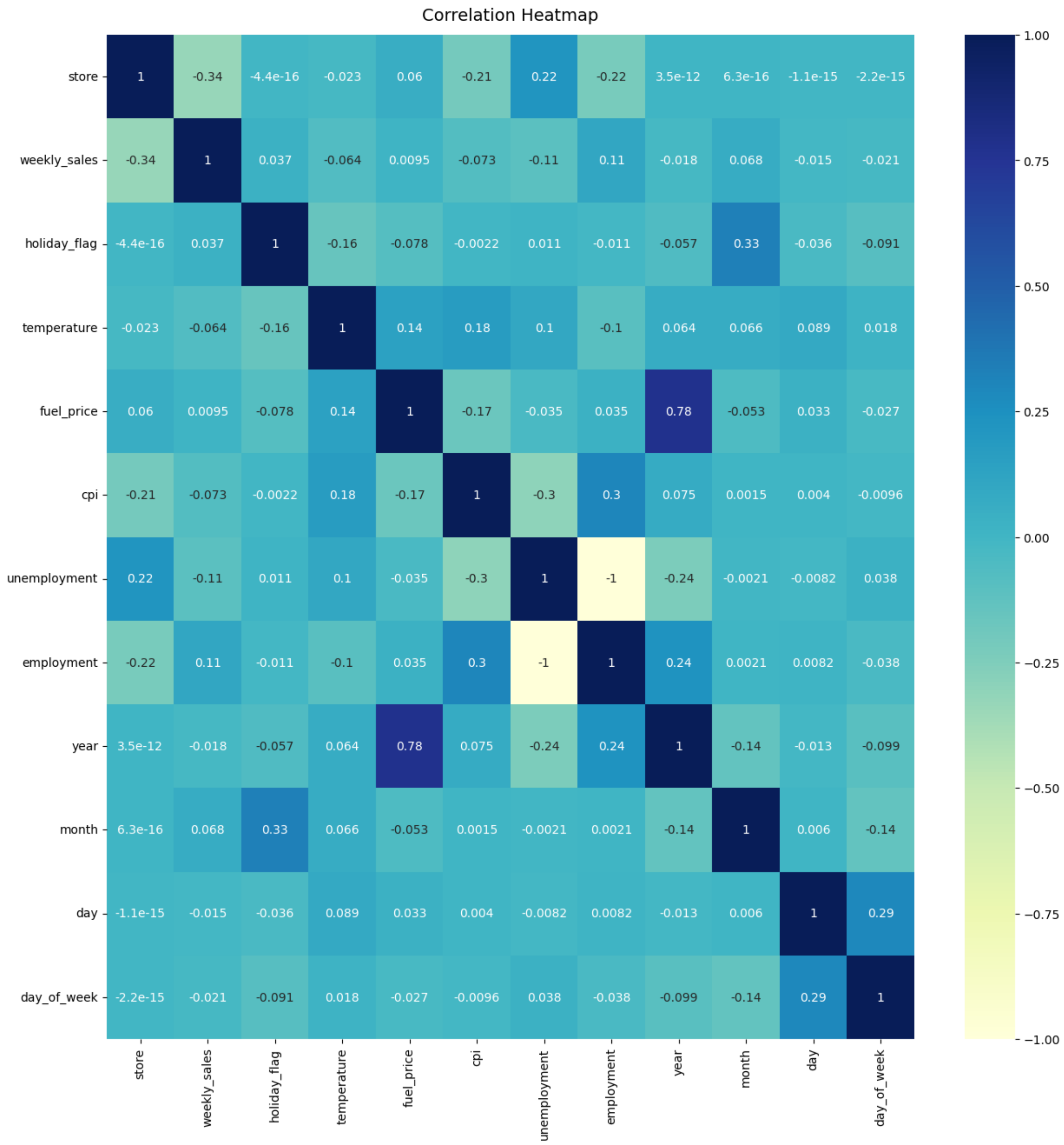
In [157…
```
fig, ax = plt.subplots(figsize=(20, 6))
sns.set_palette("bright")
sns.lineplot(x=pivot_table.index, y=pivot_table[2010]/1e6, ax=ax, label='2010')
sns.lineplot( x=pivot_table.index, y=pivot_table[2011]/1e6, ax=ax, label='2011')
sns.lineplot( x=pivot_table.index, y=pivot_table[2012]/1e6, ax=ax, 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()
```



In [158…

```
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);
```



Correlation Heatmap
store
weekly_sales
holiday_flag
temperature
fuel_price
cpi
unemployment
employment
year
month
day
day_of_week

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.

Timeseries Forecasting using Prophet

In [159…
```
# from fbprophet import Prophet #you need to install fbprophet using pip install
# fbprophet not able to install
```

In [160…
```
!pip install prophet
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheel
s/public/simple/
Requirement already satisfied: prophet in /usr/local/lib/python3.9/dist-packages
(1.1.2)
Requirement already satisfied: holidays>=0.14.2 in /usr/local/lib/python3.9/dist-p
ackages (from prophet) (0.22)
Requirement already satisfied: cmdstanpy>=1.0.4 in /usr/local/lib/python3.9/dist-p
ackages (from prophet) (1.1.0)
Requirement already satisfied: numpy>=1.15.4 in /usr/local/lib/python3.9/dist-pack
ages (from prophet) (1.22.4)
Requirement already satisfied: convertdate>=2.1.2 in /usr/local/lib/python3.9/dist
-packages (from prophet) (2.4.0)
Requirement already satisfied: tqdm>=4.36.1 in /usr/local/lib/python3.9/dist-packa
ges (from prophet) (4.65.0)
Requirement already satisfied: LunarCalendar>=0.0.9 in /usr/local/lib/python3.9/di
st-packages (from prophet) (0.0.9)
Requirement already satisfied: matplotlib>=2.0.0 in /usr/local/lib/python3.9/dist-
packages (from prophet) (3.7.1)
Requirement already satisfied: python-dateutil>=2.8.0 in /usr/local/lib/python3.9/
dist-packages (from prophet) (2.8.2)
Requirement already satisfied: pandas>=1.0.4 in /usr/local/lib/python3.9/dist-pack
ages (from prophet) (1.4.4)
Requirement already satisfied: pymeeus<=1,>=0.3.13 in /usr/local/lib/python3.9/dis
t-packages (from convertdate>=2.1.2->prophet) (0.5.12)
Requirement already satisfied: korean-lunar-calendar in /usr/local/lib/python3.9/d
ist-packages (from holidays>=0.14.2->prophet) (0.3.1)
Requirement already satisfied: hijri-converter in /usr/local/lib/python3.9/dist-pa
ckages (from holidays>=0.14.2->prophet) (2.2.4)
Requirement already satisfied: pytz in /usr/local/lib/python3.9/dist-packages (fro
m LunarCalendar>=0.0.9->prophet) (2022.7.1)
Requirement already satisfied: ephem>=3.7.5.3 in /usr/local/lib/python3.9/dist-pac
kages (from LunarCalendar>=0.0.9->prophet) (4.1.4)
Requirement already satisfied: importlib-resources>=3.2.0 in /usr/local/lib/python
3.9/dist-packages (from matplotlib>=2.0.0->prophet) (5.12.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.9/dist-
packages (from matplotlib>=2.0.0->prophet) (1.4.4)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.9/dist-p
ackages (from matplotlib>=2.0.0->prophet) (1.0.7)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.9/dist-p
ackages (from matplotlib>=2.0.0->prophet) (3.0.9)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.9/dist-packa
ges (from matplotlib>=2.0.0->prophet) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.9/dist-
packages (from matplotlib>=2.0.0->prophet) (4.39.3)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.9/dist-pa
ckages (from matplotlib>=2.0.0->prophet) (23.0)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.9/dist-pack
ages (from matplotlib>=2.0.0->prophet) (8.4.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.9/dist-packages
(from python-dateutil>=2.8.0->prophet) (1.16.0)
Requirement already satisfied: zipp>=3.1.0 in /usr/local/lib/python3.9/dist-packag
es (from importlib-resources>=3.2.0->matplotlib>=2.0.0->prophet) (3.15.0)
```

In [161…

```
from prophet import Prophet
```

In [162…

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

In [163…

```
test_data
```

Out[163]:

date	weekly_sales
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

In [164…

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

In [165…

```
test_data.head()
```

Out[165]:

date	weekly_sales	ds	y
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

In [166…

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

```
INFO:prophet:Disabling daily seasonality. Run prophet with daily_seasonality=True
to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpl_zclov7/yp4kq9ip.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpl_zclov7/pvv67e2p.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.9/dist-packages/prophet/sta
n_model/prophet_model.bin', 'random', 'seed=98667', 'data', 'file=/tmp/tmpl_zclov
7/yp4kq9ip.json', 'init=/tmp/tmpl_zclov7/pvv67e2p.json', 'output', 'file=/tmp/tmpl
_zclov7/prophet_modelexcw9lwk/prophet_model-20230410092425.csv', 'method=optimiz
e', 'algorithm=lbfgs', 'iter=10000']
09:24:25 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
09:24:26 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
```

Out[166]: <prophet.forecaster.Prophet at 0x7f30714ddf10>

In [167…
```
future = model.make_future_dataframe(periods=365)
future.tail()
```

Out[167]:

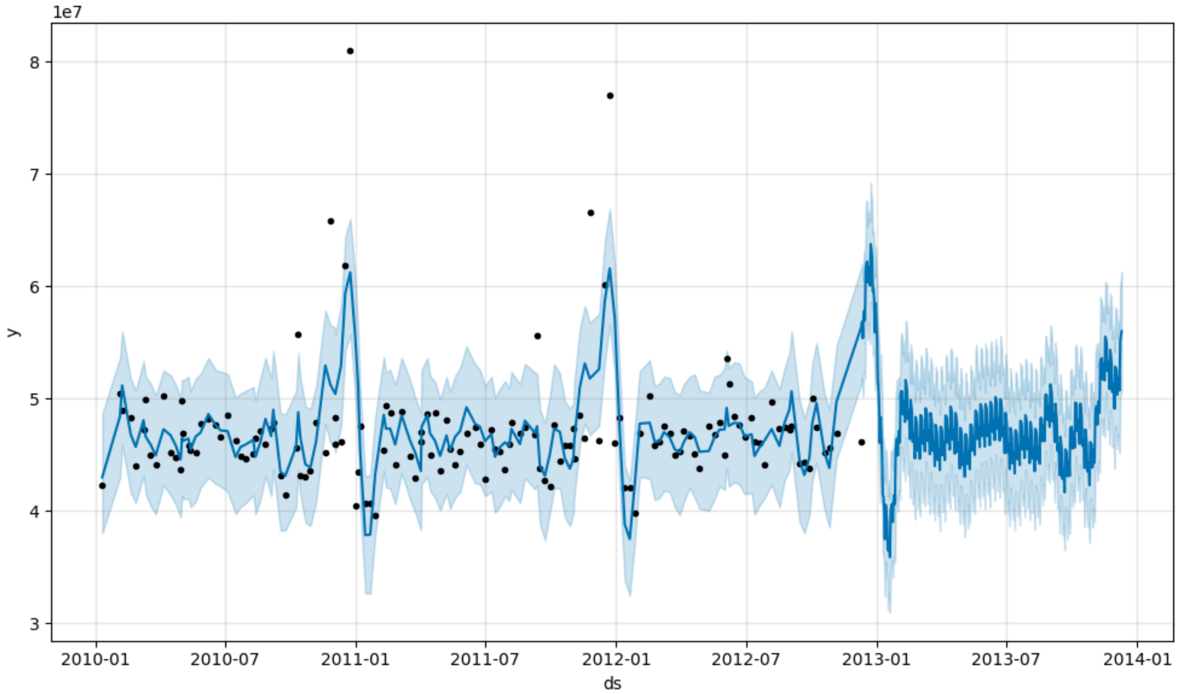
	ds
503	2013-12-06
504	2013-12-07
505	2013-12-08
506	2013-12-09
507	2013-12-10

In [168…
```
forecast = model.predict(future)
forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail()
```

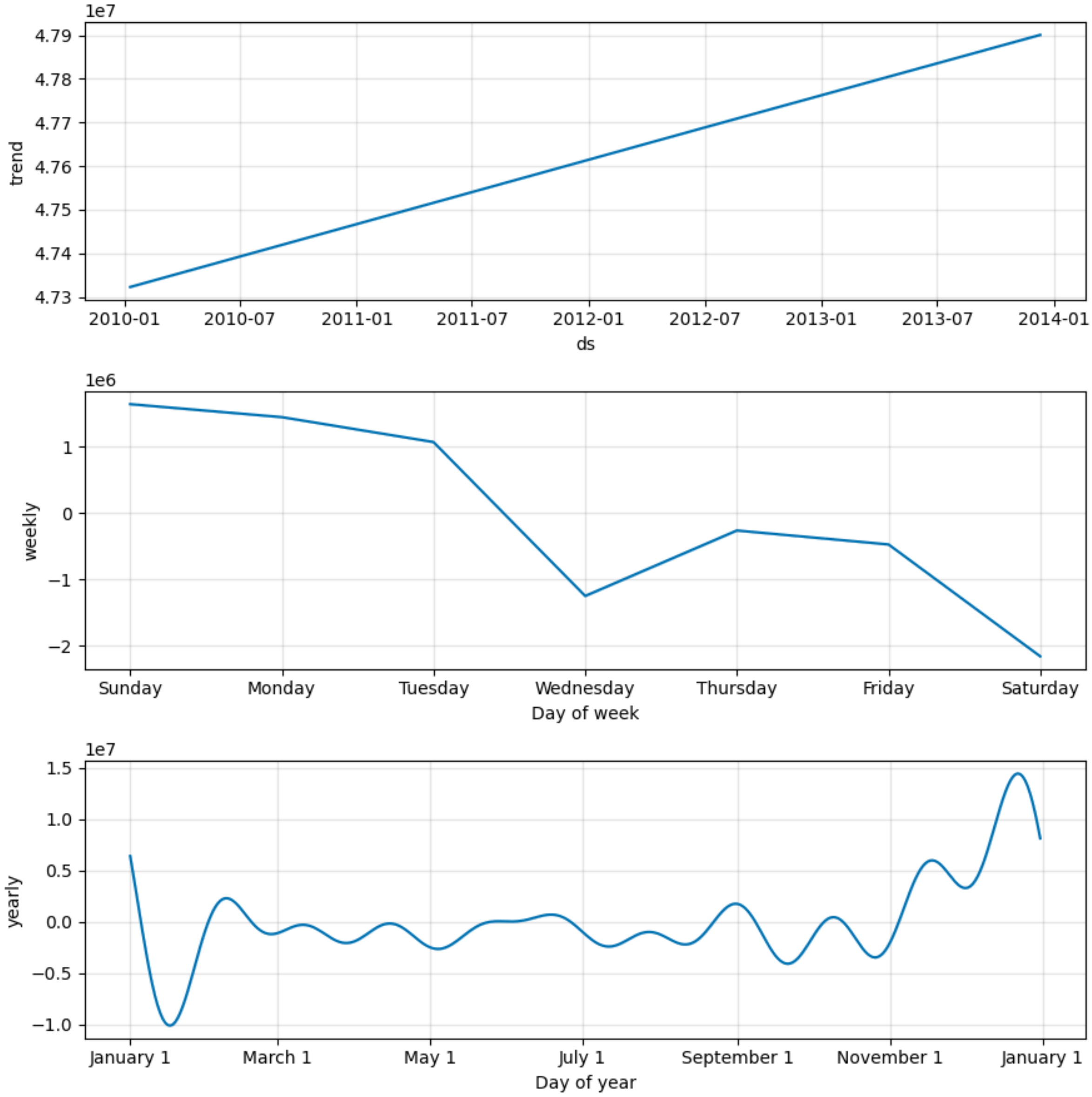
Out[168]:

	ds	yhat	yhat_lower	yhat_upper
503	2013-12-06	5.186188e+07	4.676935e+07	5.706521e+07
504	2013-12-07	5.070088e+07	4.510708e+07	5.613681e+07
505	2013-12-08	5.510082e+07	5.014708e+07	6.037101e+07
506	2013-12-09	5.559090e+07	5.033978e+07	6.052189e+07
507	2013-12-10	5.596651e+07	5.071457e+07	6.127572e+07

In [169…
```
fig1 = model.plot(forecast)
```



In [170…
```
fig2 = model.plot_components(forecast)
```



In [171…

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

In [172…

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

In [173…

```
from statsmodels.tsa.stattools import adfuller
adfuller(test_data['ds'])
```

Out[173]:

```
(22.94634042232945,
 1.0,
 0,
 142,
 {'1%': -3.477261624048995,
  '5%': -2.8821181874544233,
  '10%': -2.5777431104939494},
 8908.452193905656)
```

In [174…

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

```
Results of Dickey-Fuller Test:
Test Statistic                 22.946340
p-value                         1.000000
#lags Used                      0.000000
Number of Observations Used   142.000000
Critical Value (1%)            -3.477262
Critical Value (5%)            -2.882118
Critical Value (10%)           -2.577743
dtype: float64
```

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.

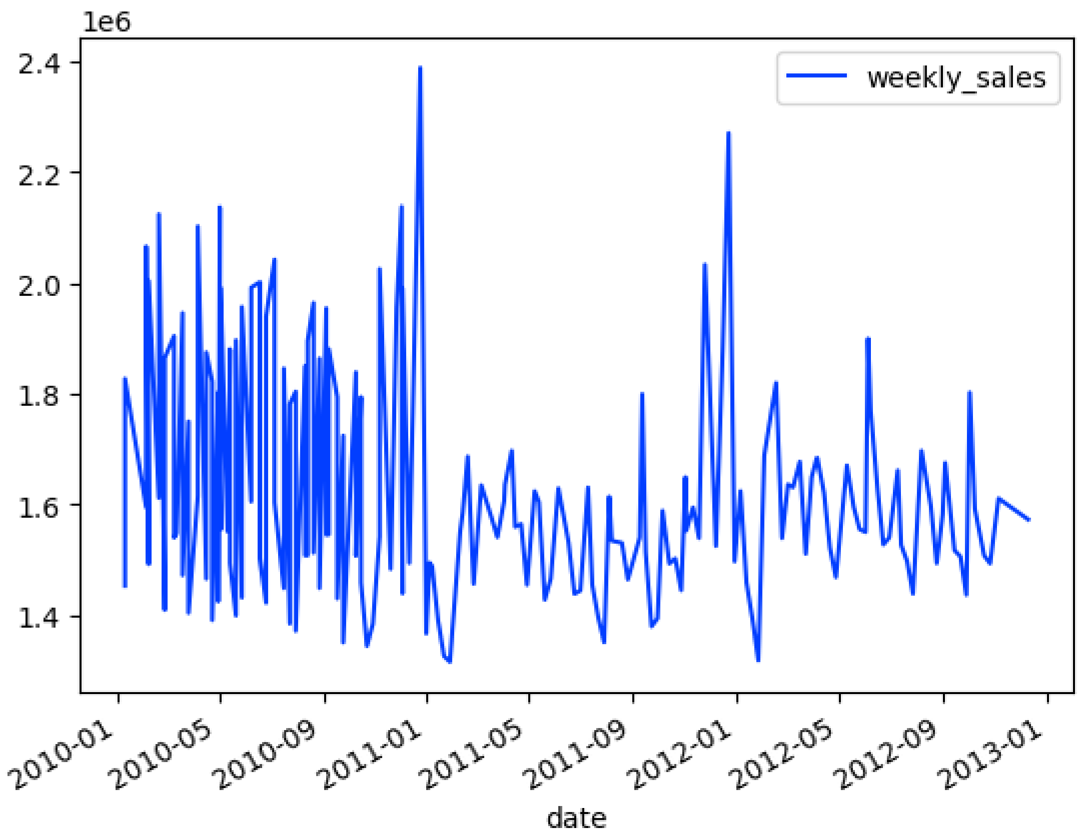
These models are capable of capturing seasonality,trends and other time-based patterns in the data, which makes them well-suited for this problem.

In [175…

```
first_six_month = data[:180]
first_six_month.plot.line(x='date', y='weekly_sales')
```

Out[175]:

```
<Axes: xlabel='date'>
```

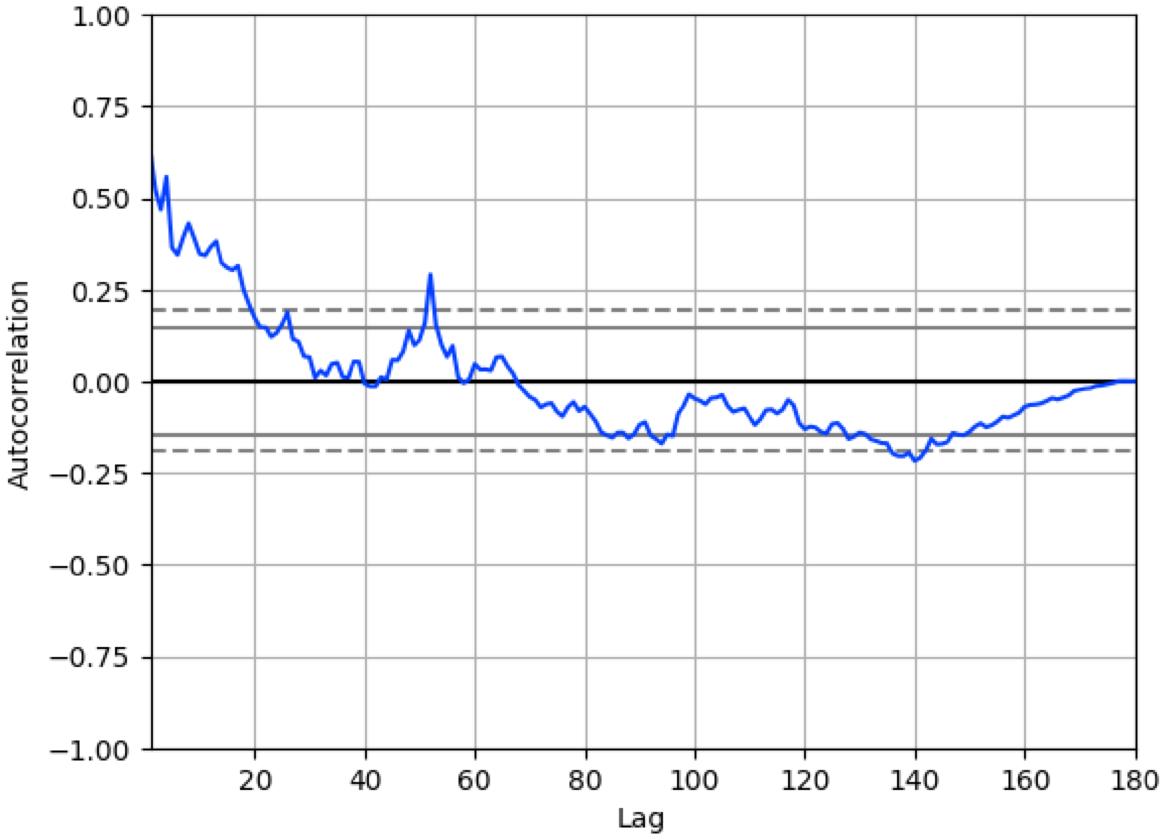


In [176…

```
from pandas.plotting import autocorrelation_plot
autocorrelation_plot(first_six_month['weekly_sales'])
```

Out[176]:

```
<Axes: xlabel='Lag', ylabel='Autocorrelation'>
```



In [177…

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

Out[177]:

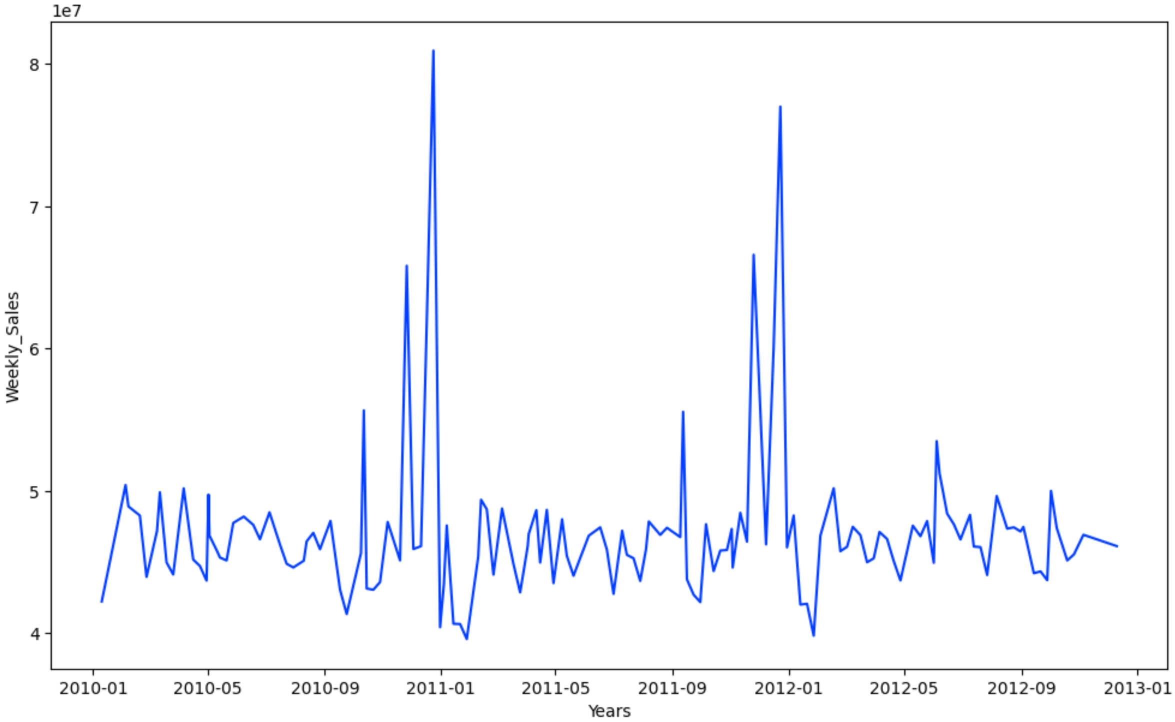
```
date
2010-01-10    42239875.87
2010-02-04    50423831.26
2010-02-07    48917484.50
2010-02-19    48276993.78
2010-02-26    43968571.13
                  ...
2012-10-08    47403451.04
2012-10-19    45122410.57
2012-10-26    45544116.29
2012-11-05    46925878.99
2012-12-10    46128514.25
Name: weekly_sales,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

In [178…

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

Out[178]:

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



1e7
8
7
6
5
4
Weekly_Sales
2010-01
2010-05
2010-09
2011-01
2011-05
2011-09
2012-01
2012-05
2012-09
2013-01
Years

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.

In [179…

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

In [180…

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

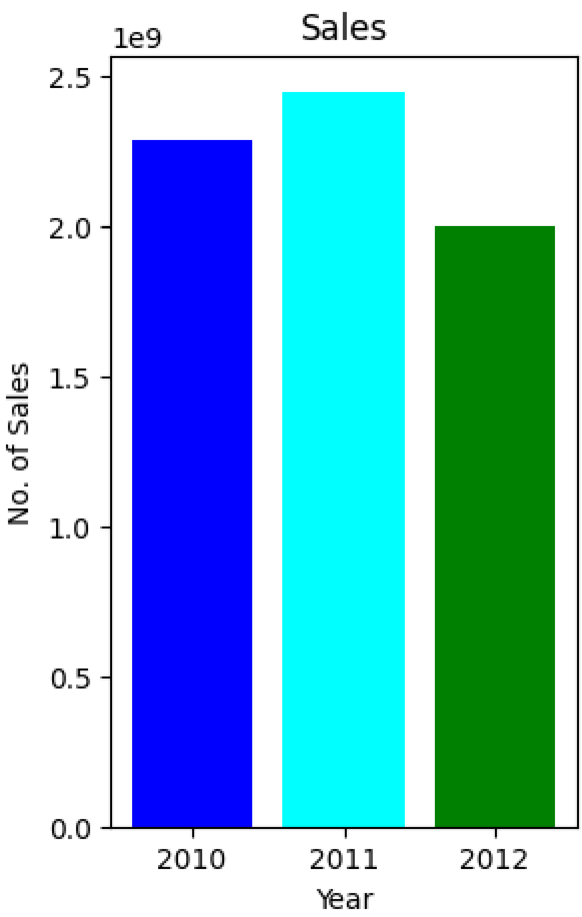
In [181…

```
# 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()
```



In [182...

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

---

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.

In [183...

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

Out[183]:

	store	date	weekly_sales	holiday_flag	temperature	fuel_price	cpi	unemployment	e
0	1	2010-05-02	1643690.90	0	42.31	2.572	211.096358	8.106	
1	1	2010-12-02	1641957.44	1	38.51	2.548	211.242170	8.106	
2	1	2010-02-19	1611968.17	0	39.93	2.514	211.289143	8.106	
3	1	2010-02-26	1409727.59	0	46.63	2.561	211.319643	8.106	
4	1	2010-05-03	1554806.68	0	46.50	2.625	211.350143	8.106	

In [184...

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

Out[184]:

```
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]
```

In [185...

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

In [186...

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

In [187...

```
from sklearn import preprocessing
from sklearn import utils

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

In [188…

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

In [189…

```
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
```

In [190…

```
def evaluate_regressors_rmses(regressors, regressor_names, X_train, y_train, X_tes
    # evaluate the models and compute their RMSE on the test data
    rmses = [evaluate_model(regressor, X_train, y_train, X_test, y_test) for regre
    # 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)
```

In [191…

```
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, lo
knn_regressor = KNeighborsRegressor(n_neighbors=5, weights='uniform')
spline_regressor = make_pipeline(PolynomialFeatures(3), LinearRegression())
```

In [192…

```
regressors = [linear_regressor, logistic_regression, polynomial_regressor, ridge_re
decision_tree_regressor, random_forest_regressor, boosted_tree_regressor, neural_ne
support_vector_regressor, knn_regressor, spline_regressor]

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

In [193…

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

```
 Table of regressors and their RMSEs
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_multilayer_perceptr
on.py:686: ConvergenceWarning:

Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't
converged yet.
```

Out[193]:

	regressor_name	rmse
0	Random Forest Regressor	313.377236
1	Decision Tree Regression	417.967196
2	Boosted Tree Regression	558.903563
3	Spline Regression	1468.104812
4	Polynomial Regression	1572.784258
5	K-Nearest Neighbour Regression	1613.757114
6	Ridge Regression	1747.785042
7	Lasso	1747.971167
8	Linear Regression	1747.991701
9	Elastic Net Regression	1772.306088
10	Neural Network Regression	1824.134778
11	Support Vector Regressor	1865.410268
12	Logistic_Regression	2207.653307

In [194…

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

```
/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/_multilayer_perceptr
on.py:686: ConvergenceWarning:

Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't
converged yet.
```

In [195…

```
# 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 [196…

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

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 :**
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/]
(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]
(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/]
(https://www.marketing91.com/sales-forecasting/)
```