Retail Analysis with Walmart Data

Abstract

The purpose of the project “Retail Analysis with Walmart Data” is to build a model which predict sales and demand accurately. There are certain events and holidays which impact the sales on each day. Sales data of 45 stores are given. In this project, we have to find the maximum sales of store, maximum standard deviation, coefficient of variation, growth rate over Q3, negative impact of holidays over sales, providing monthly and semester view of sales and building a model to predict the sales.

Introduction

The purpose of the project "Retail Analysis with Walmart Data" is to build a model which predict sales and demand accurately. There are certain events and holidays which impact the sales on each day. Sales data of 45 stores are given. In this project, we have to find the maximum sales of store, maximum standard deviation, coefficient of variation, growth rate over Q3, negative impact of holidays over sales, providing monthly and semester view of sales and building a model to predict the sales.

- By using data science, we have to perform the following tasks:
  1. Which store has maximum sales
  2. Which store has maximum standard deviation i.e., the sales vary a lot. Also, find out the coefficient of mean to standard deviation
  3. Which store/s has good quarterly growth rate in Q3'2012
  4. Some holidays have a negative impact on sales. Find out holidays which have higher sales than the mean sales in non-holiday season for all stores together
  5. Provide a monthly and semester view of sales in units and give insights
  6. For Store 1 – Build prediction models to forecast demand
     - Linear Regression – Utilize variables like date and restructure dates as 1 for 5 Feb 2010 (starting from the earliest date in order). Hypothesize if CPI, unemployment, and fuel price have any impact on sales.
     - Change dates into days by creating new variable.

The data is provided to us in the form of a CSV file.

Data Science

Data science is the domain of study that deals with vast volumes of data using modern tools and techniques to find unseen patterns, derive meaningful information, and make business decisions. Data science uses complex machine learning algorithms to build predictive models.

The data used for analysis can come from many different sources and presented in various formats.

Data science is an essential part of many industries today, given the massive amounts of data that are produced, and is one of the most debated topics in IT circles. Its popularity has grown over the years, and companies have started implementing data science techniques to grow their business and increase customer satisfaction

The Data Science Lifecycle:

Data science's lifecycle consists of five distinct stages, each with its own tasks:

1. Capture: Data Acquisition, Data Entry, Signal Reception, Data Extraction. This stage involves gathering raw structured and unstructured data.
2. Maintain: Data Warehousing, Data Cleansing, Data Staging, Data Processing, Data Architecture. This stage covers taking the raw data and putting it in a form that can be used.
3. Process: Data Mining, Clustering/Classification, Data Modeling, Data Summarization. Data scientists take the prepared data and examine its patterns, ranges, and biases to determine how useful it will be in predictive analysis.
4. Analyze: Exploratory/Confirmatory, Predictive Analysis, Regression, Text Mining, Qualitative Analysis. Here is the real meat of the lifecycle. This stage involves performing the various analyses on the data.
5. Communicate: Data Reporting, Data Visualization, Business Intelligence, Decision Making. In this final step, analysts prepare the analyses in easily readable forms such as charts, graphs, and reports.

Prerequisites for Data Science:

1. Machine Learning
2. Modeling
3. Statistics
4. Programming
5. Databases

Python for Data Science

Python is open source, interpreted, high level language and provides great approach for object-oriented programming. It is one of the best language used by data scientist for various data science projects/application. Python provide great functionality to deal with mathematics, statistics and scientific function. It provides great libraries to deals with data science application.

One of the main reasons why Python is widely used in the scientific and research communities is because of its ease of use and simple syntax which makes it easy to adapt for people who do not have an engineering background. It is also more suited for quick prototyping.

In terms of application areas, ML scientists prefer Python as well. When it comes to areas like building fraud detection algorithms and network security, developers leaned towards Java, while for applications like natural language processing (NLP) and sentiment analysis, developers opted for Python, because it provides large collection of libraries that help to solve complex business problem easily, build strong system and data application.

Commonly used Python libraries for Data Science:

1. **Numpy**: Numpy is Python library that provides mathematical function to handle large dimension array. It provides various method/function for Array, Metrics, and linear algebra.
2. **Pandas**: Pandas is one of the most popular Python library for data manipulation and analysis. Pandas provide useful functions to manipulate large amount of structured data. Pandas provide easiest method to perform analysis. It provides large data structures and manipulating numerical tables and time series data. Pandas is a perfect tool for data wrangling. Pandas is designed for quick and easy data manipulation, aggregation, and visualization.
3. **Matplotlib**: Matplotlib is another useful Python library for Data Visualization. Descriptive analysis and visualizing data is very important for any organization. Matplotlib provides various method to Visualize data in more effective way. Matplotlib allows to quickly make line graphs, pie charts, histograms, and other professional grade figures.
4. **Scikit – Learn**: Sklearn is Python library for machine learning. Sklearn provides various algorithms and functions that are used in machine learning. Sklearn is built on NumPy, SciPy, and matplotlib. Sklearn provides easy and simple tools for data mining and data analysis. It

provides a set of common machine learning algorithms to users through a consistent interface. Scikit-Learn helps to quickly implement popular algorithms on datasets and solve real-world problems.

NumPy

NumPy stands for Numerical Python. NumPy is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices. It is an open source project and you can use it freely.

In Python we have lists that serve the purpose of arrays, but they are slow to process. NumPy aims to provide an array object that is up to 50x faster than traditional Python lists. The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy. Arrays are very frequently used in data science, where speed and resources are very important.

NumPy arrays are stored at one continuous place in memory unlike lists, so processes can access and manipulate them very efficiently. This behavior is called locality of reference in computer science. This is the main reason why NumPy is faster than lists.

Pandas

Pandas is an open-source library that is built on top of NumPy library. It is a Python package that offers various data structures and operations for manipulating numerical data and time series. It is mainly popular for importing and analyzing data much easier. Pandas is fast and it has high-performance & productivity for users.

- **Advantages**
  1. Fast and efficient for manipulating and analyzing data.
  2. Data from different file objects can be loaded.

3. Easy handling of missing data (represented as NaN) in floating point as well as non-floating point data
4. Size mutability: columns can be inserted and deleted from DataFrame and higher dimensional objects
5. Data set merging and joining.
6. Flexible reshaping and pivoting of data sets
7. Provides time-series functionality.
8. Powerful group by functionality for performing split-apply-combine operations on datasets.

➢ **Series**
  - Pandas Series is a one-dimensional labelled array capable of holding data of any type (integer, string, float, python objects, etc.). The axis labels are collectively called indexes. Pandas Series is nothing but a column in an excel sheet. Labels need not be unique but must be a hash able type. The object supports both integer and label-based indexing and provides a host of methods for performing operations involving the index.

➢ **DataFrame**
  - Pandas DataFrame is a two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns). A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns. Pandas DataFrame consists of three principal components, the data, rows, and columns.

Matplotlib

Matplotlib is easy to use and an amazing visualizing library in Python. It is built on NumPy arrays and designed to work with the broader SciPy stack and consists of several plots like line, bar, scatter, histogram, pie charts, box plot etc.

- **Line Graph**: Line Chart is used to represent a relationship between two data X and Y on a different axis. It is plotted using the plot() function.

- **Bar Graph**: A bar plot or bar chart is a graph that represents the category of data with rectangular bars with lengths and heights that is proportional to the values which they represent. The bar plots can be plotted horizontally or vertically. A bar chart describes the comparisons between the discrete categories. It is created using the bar() method.

- **Histogram:** A histogram is basically used to represent data in the form of some groups. It is a type of bar plot where the X-axis represents the bin ranges while the Y-axis gives information about frequency. To create a histogram, the first step is to create a bin of the ranges, then distribute the whole range of the values into a series of intervals, and count the values which fall into each of the intervals. Bins are clearly identified as consecutive, non-overlapping intervals of variables. The hist() function is used to compute and create histogram of x.

- **Scatter Plot:** Scatter plots are used to observe relationship between variables and uses dots to represent the relationship between them. The scatter() method in the matplotlib library is used to draw a scatter plot.

- **Pie Chart:** A Pie Chart is a circular statistical plot that can display only one series of data. The area of the chart is the total percentage of the given data. The area of slices of the pie represents the percentage of the parts of the data. The slices of pie are called wedges. The area of the wedge is determined by the length of the arc of the wedge. It can be created using the pie() method.

- **Box Plot:** A Box Plot is also known as Whisker plot is created to display the summary of the set of data values having properties like minimum, first quartile, median, third quartile and maximum. In the box plot, a box is created from the first quartile to the third quartile, a vertical line is also there which goes through the box at the median. Here x-axis denotes the data to be plotted while the y-axis shows the frequency distribution.

Scikit-Learn

Scikit-Learn is an open-source Python library that implements a range of machine learning, pre-processing, cross-validation, and visualization algorithms using a unified interface.

- **Important features of scikit-learn:**
  1. Simple and efficient tools for data mining and data analysis. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means, etc.
  2. Accessible to everybody and reusable in various contexts.
  3. Built on the top of NumPy, SciPy, and matplotlib.
  4. Open source, commercially usable – BSD license.

Scikit-Learn is used to build a ML model.

- **Steps for building a ML model**
  1. Load the dataset and divide it into Features and Response
     - **Features**: (also known as predictors, inputs, or attributes) they are simply the variables of our data. They can be more than one and hence represented by a feature matrix ('X' is a common notation to represent feature matrix). A list of all the feature names is termed feature names.
     - **Response:** (also known as the target, label, or output) This is the output variable depending on the feature variables. We generally have a single response column and it is represented by a response vector ('y' is a common notation to represent response vector). All the possible values taken by a response vector are termed target names.

  2. **Split the dataset into training and testing datasets**
     - Split the dataset into two pieces: a training set and a testing set.
     - Train the model on the training set.

    - Test the model on the testing set, and evaluate how well our model did.

3. **Training the model**
    - Use ML algorithms to fit, predict and find accuracy.

Source Code

Import the required libraries, load the dataset and explore it

```
# Import required libraries
import numpy as np
import pandas as pd
from matplotlib import dates
from datetime import datetime
from matplotlib import pyplot as plt
import seaborn as sns
%matplotlib inline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score,mean_squared_error

# Load the dataset
df = pd.read_csv('Walmart_Store_sales.csv')

# Check the first 5 records
df.head()
# Find out the number of rows and columns
df.shape
# See more info about the dataset
df.info()
# The datatype of Date column is object. Convert it to datetime
```

df['Date'] = pd.to_datetime(df['Date'])

df.info()

# Check for null values in the dataset

df.isnull().sum()

Task 1: Which store has the maximum sales

# Group the dataset according to the stores and find the sum of weekly sales and sort them in descending order

max_store_sales =
df.groupby('Store')['Weekly_Sales'].sum().sort_values(ascending=False)

# Convert the series into a dataframe and reset its index

max_store_sales = pd.DataFrame(max_store_sales)

max_store_sales.reset_index(inplace=True)

max_store_sales

# Plot the store and sales on a bar graph to find the store with maximum and minimum sales

# Use styles for better presentation

plt.style.use('fivethirtyeight')

# Set the figure size to 20:8 to present it

plt.figure(figsize=(20, 8))

x = max_store_sales['Store']

y = max_store_sales['Weekly_Sales']

# Set color of the bar graph depending on the maximum and minimum sales

color = ['lightsteelblue' if (x < max(max_store_sales['Weekly_Sales']) and x >
min(max_store_sales['Weekly_Sales']))

    else 'midnightblue' for x in max_store_sales['Weekly_Sales']]

```
# Plot the graph
plt.bar(x, y, color=color, width=0.6)
# Set the properties of the graph and display it
plt.title('Store with maximum sales')
plt.xlabel('Store')
plt.ylabel('Sales')
plt.xticks(ticks=x, labels=x)
plt.tight_layout()
plt.show()
```

Task 2: Which store has maximum standard deviation i.e. the sales vary a lot. Also, find out the coefficient of mean to standard deviation

```
# Group the dataset according to the stores and find the sum of weekly sales.
# Appply std() & mean() and sort them in descending order according to std()
max_store_std = df.groupby('Store')['Weekly_Sales'].agg(['std',
'mean']).sort_values(by='std',ascending=False)


# Convert the result into dataset, reset index and rename columns
max_store_std = pd.DataFrame(max_store_std)
max_store_std.reset_index(inplace=True)
max_store_std.rename(columns={"Store":'Store', 'std':'Standard_Deviation',
'mean':'Mean'}, inplace=True)
max_store_std


# Print the store with maximum std()
print(f"Store #{max_store_std.loc[0, 'Store']} \
has the maximum standard deviation of {max_store_std.loc[0,
'Standard_Deviation']:.2f}")
```

```
# Plot the store and sales on a bar graph to find the store with maximum and minimum sales

# Use styles for better presentation

plt.style.use('classic')

# Set the figure size to 20:8 to present it

plt.figure(figsize=(20, 8))

x = max_store_std['Store']

y = max_store_std['Standard_Deviation']

# Set color of the bar graph depending on the maximum and minimum sales

color = ['#B8405E' if (x < max(max_store_std['Standard_Deviation']))
         else '#000957' for x in max_store_std['Standard_Deviation']]


# Plot the graph

plt.bar(x, y, color=color, width=0.6)

# Set the properties of the graph and display it

plt.title('Store with maximum standard deviation')

plt.xlabel('Store')

plt.ylabel('Standard Deviation')

plt.xticks(ticks=x, labels=x)

plt.tight_layout()

plt.show()


# Apply filter to find all the data related to Store #14

filt = df[df['Store'] == 14]

filt


# Plot the sales of Store #14 on a histogram

plt.style.use('fivethirtyeight')
plt.figure(figsize=(20,8))

plt.hist(filt['Weekly_Sales'], bins=100, label='Sales')

plt.xlabel('Sales')

plt.title('Sales Distribution of Store #14')

plt.ylabel('Value')

plt.legend()

plt.tight_layout()

plt.show()


# Find the coefficient of variation or mean to standard deviation

# Do this by grouping the data according to store and finding the weekly sales

grp = df.groupby('Store')['Weekly_Sales']


# Apply the function for coefficient of variation, i.e, standard_deviation/mean * 
100

coef_variation = (grp.std()/grp.mean()) * 100


# Convert the result into dataset, reset index, rename columns and sort 
according to coefficient of variation

coef_variation = pd.DataFrame(coef_variation)

coef_variation.reset_index(inplace=True)

coef_variation.rename(columns={'Store':'Store', 
'Weekly_Sales':'Coefficient_of_Variation'}, inplace=True)

coef_variation.sort_values(by='Coefficient_of_Variation', ascending=False, 
inplace=True)

coef_variation


# Print the store with highest coefficient of variation and print by how much
```

```
print(f"Store #{coef_variation.loc[34, 'Store']} has the highest Coefficient of Variation, i.e,\n
Coefficient of mean to Standard Deviation of {coef_variation.loc[34, 'Coefficient_of_Variation']:.2f}"')


# Apply filter to find all the data of Store #35
filt = df[df['Store'] == 35]
filt


# Plot a histogram of weekly sales of Store #35
plt.style.use('fivethirtyeight')
plt.figure(figsize=(20,8))
plt.hist(filt['Weekly_Sales'], bins=100, label='Sales')
plt.xlabel('Sales')
plt.title('Sales Distribution of Store #35')
plt.ylabel('Value')
plt.legend()
plt.tight_layout()
plt.show()


# Plot a bar graph to find the coefficient of variation of all the stores and veriy if Store #35 is the highest
plt.style.use('fivethirtyeight')
plt.figure(figsize=(20,8))
# Se colors according to the max coefficient of variation
clr = ['#7A0BC0' if x == max(coef_variation['Coefficient_of_Variation']) else '#398AB9' for x in coef_variation['Coefficient_of_Variation']]
plt.bar(coef_variation['Store'], coef_variation['Coefficient_of_Variation'],
    label='Coefficient of Variation', color=clr, width=0.6)
```

```
plt.xlabel('Store')
plt.ylabel('Coefficient of Variation')
plt.xticks(ticks=coef_variation['Store'], label=coef_variation['Store'])
plt.title('Coefficient of Variation of all the Stores')
plt.legend()
plt.tight_layout()
plt.show()
```

Task 3: Which store/s has good quarterly growth rate in Q3’2012?

```
# Find count, mean, std, min, 25%, 50%, 75% and max in the dataset
df.describe()
# Set the time frame for Q2 and Q3. Groupt them by store and find sum of their
weekly sales
q2 = df[(df['Date'] >= '2012-04-01') & (df['Date'] <= '2012-06-
30')].groupby('Store')['Weekly_Sales'].sum()
q3 = df[(df['Date'] >= '2012-07-01') & (df['Date'] <= '2012-09-
30')].groupby('Store')['Weekly_Sales'].sum()

# Conert them into dataframes, reset index and rename columns
q2 = pd.DataFrame(q2)
q3 = pd.DataFrame(q3)
q2.reset_index(inplace=True)
q3.reset_index(inplace=True)
q2.rename(columns={'Store':'Store', 'Weekly_Sales':'Q2'}, inplace=True)
q3.rename(columns={'Store':'Store', 'Weekly_Sales':'Q3'}, inplace=True)

# Apply filter to find max of Q3
filt = (q3[q3['Q3'] == max(q3['Q3'])])
```

```
filt.reset_index(inplace=True)
```

```
# Print the store with max of Q3 along with its growth rate

print(f"Store #{filt.loc[0, 'Store']} has the maximum growth rate in Q3'2012 with {filt.loc[0, 'Q3']}$")
```

```
# Plot a bar graph to see the growth rate of sales over Q3'2012

plt.style.use('fivethirtyeight')

plt.figure(figsize=(20,10))

plt.bar(q2['Store'], q2['Q2'], label='Q2\'2012', color='#B83B5E', width=0.5)

plt.bar(q2['Store'], q3['Q3'], label='Q3\'2012', color='#03045E', alpha=.9, width=0.5)

plt.xlabel('Store')

plt.xticks(ticks=q2['Store'], label=q2['Store'])

plt.ylabel('Sales Over Q2 & Q3')

plt.title('Growth Rate of Sales over Q3\'2012')

plt.tight_layout()

plt.legend()

plt.show()
```

Task 4: Some holidays have a negative impact on sales. Find out holidays which have higher sales than the mean sales in non-holiday season for all stores together

```
# Find total sales by grouping by date and finding sum of weekly sales

total_sales = df.groupby('Date')['Weekly_Sales'].sum().reset_index()
```

```
# Holiday dates are provided

super_bowl = ['2010-02-12', '2011-02-11', '2012-02-10', '2013-02-13']
```

```
labour_day = ['2010-09-10', '2011-09-09', '2012-09-07', '2013-09-06']
thanksgiving_day = ['2010-11-26', '2011-11-25', '2012-11-23', '2013-11-29']
christmas = ['2010-12-31', '2011-12-30', '2012-12-28', '2013-12-27']

# Write a function that displays line plot and also highlightes the holidays and shows their sales on that particular day

def plot_line(df,holiday_dates,label):
    plt.style.use('classic')
    fig, ax = plt.subplots(figsize = (15,5))
    ax.plot(df['Date'],df['Weekly_Sales'],label=label)

    for day in holiday_dates:
        day = datetime.strptime(day, '%Y-%m-%d')
        plt.axvline(x=day, linestyle='--', c='r'

    plt.title(label)
    x_dates = df['Date'].dt.strftime('%Y-%m-%d').sort_values().unique()
    xfmt = dates.DateFormatter('%d-%m-%y')
    ax.xaxis.set_major_formatter(xfmt)
    ax.xaxis.set_major_locator(dates.DayLocator(1))
    plt.gcf().autofmt_xdate(rotation=90)
    plt.grid()
    plt.show()

# Use the function plot_line to find the position of sales on Super Bowl holiday
plot_line(total_sales, super_bowl, 'Super Bowl')
# Use the function plot_line to find the position of sales on labour Day holiday
```

```
plot_line(total_sales, labour_day, 'Labour Day')

# Use the function plot_line to find the position of sales on ThanksGiving Day holiday

plot_line(total_sales, thanksgiving_day, 'ThanksGiving')

# Use the function plot_line to find the position of sales on Christmas holiday

plot_line(total_sales, christmas, 'Christmas')


# Find the total sales on Super Bowl

a=df[df['Date']== '12-02-2010']['Weekly_Sales'].sum()

b=df[df['Date']== '11-02-2011']['Weekly_Sales'].sum()

c=df[df['Date']== '10-02-2012']['Weekly_Sales'].sum()

d=df[df['Date']== '08-02-2013']['Weekly_Sales'].sum()

super_bowl_sale = a+b+c+d


# Find the total sales on Labour Day

a1=df[df['Date']== '10-09-2010']['Weekly_Sales'].sum()

b1=df[df['Date']== '09-09-2011']['Weekly_Sales'].sum()

c1=df[df['Date']== '07-09-2012']['Weekly_Sales'].sum()

d1=df[df['Date']== '06-09-2013']['Weekly_Sales'].sum()

labour_day_sale = a1+b1+c1+d1


# Find the total sales on ThanksGiving Day

a2=df[df['Date']== '26-11-2010']['Weekly_Sales'].sum()

b2=df[df['Date']== '25-11-2011']['Weekly_Sales'].sum()

c2=df[df['Date']== '23-11-2012']['Weekly_Sales'].sum()

d2=df[df['Date']== '29-11-2013']['Weekly_Sales'].sum()

thanks_giving_sale = a2+b2+c2+d2
```

```
# Find the total sales on Christmas
a3=df[df['Date']== '31-12-2010']['Weekly_Sales'].sum()
b3=df[df['Date']== '30-12-2011']['Weekly_Sales'].sum()
c3=df[df['Date']== '28-12-2012']['Weekly_Sales'].sum()
d3=df[df['Date']== '27-12-2013']['Weekly_Sales'].sum()
christmas = a3+b3+c3+d3

print('super_bowl_sale : ',super_bowl_sale)
print('labour_day_sale : ',labour_day_sale)
print('thanks_giving_sale : ',thanks_giving_sale)
print('christmas : ',christmas)

# Plot a bar graph to compare different holidays
x_list = ['Super Bowl', 'Labour Day', 'ThanksGiving Day', 'Christmas']
y_list = [super_bowl_sale, labour_day_sale, thanks_giving_sale, christmas]

plt.style.use('classic')
plt.figure(figsize=(10,5))
plt.bar(x_list, y_list, label='Sales', width=0.3)
plt.xlabel('Holidays')
plt.ylabel("Sales")
plt.title('Holiday Sales Comparision')
plt.legend()
plt.tight_layout()
plt.show()

# Find the non-holiday sales mean of all the stores
```

```
non_holiday_sales_mean = df[df['Holiday_Flag'] == 0]['Weekly_Sales'].mean()
non_holiday_sales_mean

# Check if holiday sales are greater than non-holiday sales mean of all the stores
print(super_bowl_sale > non_holiday_sales_mean)
print(labour_day_sale > non_holiday_sales_mean)
print(thanks_giving_sale > non_holiday_sales_mean)
print(christmas > non_holiday_sales_mean)

# Compare holidays with non-holiday sales mean of all the stores by plotting them on a bar graph
x_list.append('Mean Non-Holidays')
y_list.append(non_holiday_sales_mean)
plt.style.use('seaborn')
plt.figure(figsize=(6,3))
plt.bar(x_list, y_list, label='Sales', width=0.3)
plt.xlabel('Holidays')
plt.ylabel("Sales")
plt.title('Holidays Vs Mean Non-Holiday Sales')
plt.legend()
plt.tight_layout()
plt.show()
```

Task 5: Provide a monthly and semester view of sales in units and give insights

```
# Create 3 new columns for day, month and year
df['Day'] = df['Date'].dt.day
```

```
df['Month'] = df['Date'].dt.month

df['Year'] = df['Date'].dt.year

df.head()


# Find the number of rows and columns in the dataframe

df.shape


# Monthly view of sales for each years

# Plot them on a bar graph

plt.style.use('classic')

fig, (ax1, ax2, ax3) = plt.subplots(nrows=3, ncols=1, sharex=True,
sharey=True)

fig.set_size_inches(10, 6)


ax1.bar(df[df['Year']==2010]["Month"],df[df['Year']==2010]["Weekly_Sales"],
color='#5D8BF4')

ax1.set_yscale("log")

ax1.set_title("Monthly view of sales in 2010, 2011, 2012 Respectlively")


ax2.bar(df[df['Year']==2011]["Month"],df[df['Year']==2011]["Weekly_Sales"],
color='#E04DB0')

ax2.set_ylabel("Weekly Sales")


ax3.bar(df[df['Year']==2012]["Month"],df[df['Year']==2012]["Weekly_Sales"],
color='#8BDB81')

ax3.set_xlabel("Months")

plt.tight_layout()

plt.show()

# Find the semester sales for each years
```

sem1_2010 = df[(df['Date'] >= '2010-01-01') & (df['Date'] <= '2010-06-30')].groupby('Date')['Weekly_Sales'].sum()

sem2_2010 = df[(df['Date'] >= '2010-07-01') & (df['Date'] <= '2010-12-31')].groupby('Date')['Weekly_Sales'].sum()

sem1_2010 = pd.DataFrame(sem1_2010)['Weekly_Sales'].sum()

sem2_2010 = pd.DataFrame(sem2_2010)['Weekly_Sales'].sum()

sem1_2011 = df[(df['Date'] >= '2011-01-01') & (df['Date'] <= '2011-06-30')].groupby('Date')['Weekly_Sales'].sum()

sem2_2011 = df[(df['Date'] >= '2011-07-01') & (df['Date'] <= '2011-12-31')].groupby('Date')['Weekly_Sales'].sum()

sem1_2011 = pd.DataFrame(sem1_2011)['Weekly_Sales'].sum()

sem2_2011 = pd.DataFrame(sem2_2011)['Weekly_Sales'].sum()

sem1_2012 = df[(df['Date'] >= '2012-01-01') & (df['Date'] <= '2012-06-30')].groupby('Date')['Weekly_Sales'].sum()

sem2_2012 = df[(df['Date'] >= '2012-07-01') & (df['Date'] <= '2012-12-31')].groupby('Date')['Weekly_Sales'].sum()

sem1_2012 = pd.DataFrame(sem1_2012)['Weekly_Sales'].sum()

sem2_2012 = pd.DataFrame(sem2_2012)['Weekly_Sales'].sum()

# Plot a pie chart to view the difference in the semesters of all the years

slices_2010 = [sem1_2010, sem2_2010]

slices_2011 = [sem1_2011, sem2_2011]

slices_2012 = [sem1_2012, sem2_2012]

labels = ['Semester 1', 'Semester 2']

plt.style.use('seaborn')

```
fig, (ax1, ax2, ax3) = plt.subplots(nrows=1, ncols=3)
fig.set_size_inches(11, 11)

ax1.pie(slices_2010, labels=labels, wedgeprops={'edgecolor':'black'},
autopct='%1.1f%%')
ax1.set_title("Semester view of Sales - 2010")

ax2.pie(slices_2011, labels=labels, wedgeprops={'edgecolor':'black'},
autopct='%1.1f%%', colors=['#F76E11', '#8A39E1'])
ax2.set_title("Semester view of Sales - 2011")

ax3.pie(slices_2012, labels=labels, wedgeprops={'edgecolor':'black'},
autopct='%1.1f%%', colors=['#FFAB76', '#24A19C'])
ax3.set_title("Semester view of Sales - 2012")

plt.tight_layout()
plt.show()
```

Task 6: For Store #1 - Build prediction models to forecast demand

- Linear Regression - Utilize variables like date and restructure dates. Hypothesize if CPI, unemployment, and fuel price have any impact on sales.
- Change dates into days by creating new variable.

```
# Create a new column to find the day of the week
df['Day_Of_Week'] = df['Date'].dt.dayofweek

# View the first 5 records
df.head()
```

```
# Convert the categorical data into dummy variables and join it to the dataframe
sample = pd.get_dummies(df["Store"])

# Drop the store column
df = df.drop('Store',axis = 1)
df = df.join(sample)

# Standardize features by removing the mean and scaling to unit variance.
# Perform standardization for Temperature, Fuel_Price and Unemployment
scaler = StandardScaler()
scale_temp = scaler.fit_transform(df[['Temperature']])
df['Temperature'] = scale_temp
scale_fuel = scaler.fit_transform(df[['Fuel_Price']])
df['Fuel_Price'] = scale_fuel
scale_unemployment = scaler.fit_transform(df[['Unemployment']])
df['Unemployment'] = scale_unemployment

# Plot Temperature, Fuel_Price, CPI & Unemployment on a boxplot to find the
outliers
plt.style.use('seaborn')
fig, (ax1, ax2, ax3, ax4) = plt.subplots(nrows=1, ncols=4)
fig.set_size_inches(15, 5)

ax1.boxplot(df['Temperature'])
ax1.set_xlabel('Temperature')

ax2.boxplot(df['Fuel_Price'])
```

```
ax2.set_xlabel('Fuel Price')

ax3.boxplot(df['CPI'])
ax3.set_xlabel('CPI')

ax4.boxplot(df['Unemployment'])
ax4.set_xlabel('Unemployment')

fig.suptitle('Checking for outliers - Temperature, Fuel Price, CPI,
Unemployment', fontsize=15)
plt.tight_layout()
plt.show()

# Remove the outliers
df = df[(df['Unemployment'] > -1.8) & (df['Unemployment'] < 1.5) &
(df['Temperature'] > -2.9)]

# Check if the outliers are removed by plotting a boxplot
plt.style.use('seaborn')
fig, (ax1, ax2, ax3, ax4) = plt.subplots(nrows=1, ncols=4)
fig.set_size_inches(15, 5)

ax1.boxplot(df['Temperature'])
ax1.set_xlabel('Temperature')

ax2.boxplot(df['Fuel_Price'])
ax2.set_xlabel('Fuel Price')
```

```
ax3.boxplot(df['CPI'])
ax3.set_xlabel('CPI')

ax4.boxplot(df['Unemployment'])
ax4.set_xlabel('Unemployment')

fig.suptitle('Checking for outliers - Temperature, Fuel Price, CPI,
Unemployment', fontsize=15)
plt.tight_layout()
plt.show()

# Create features and target for building a model
X_feature = df.drop(columns=['Weekly_Sales','Date'],axis=1)
Y_target = df['Weekly_Sales']

# Split the data into testing and training datasets with 20%-80% testing and
training datasets respectively
x_train, x_test,y_train, y_test = train_test_split(
   X_feature, Y_target, test_size=0.2, train_size=0.8, random_state=1
)

# Find the shape to check if testing and training datasets are properly split
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)

# Create an object for linear regression
```

```
model = LinearRegression()

# Fit the training dataset into the linear regression
model.fit(x_train,y_train)

# Predict the outcome based on the testing dataset of features
y_prediction = model.predict(x_test)

# Aquire the accuracy and Mean Square Error
r2 = r2_score(y_test,y_prediction)
mse = mean_squared_error(y_test,y_prediction)
accuracy = round(r2*100,2)
print(f'Accuracy is : {accuracy} %')
print('Mean Square Error :',mse)

# Plot the target testing dataset and the predictions on a regplot
plt.style.use('seaborn')
plt.figure(figsize=(10, 8))
sns.regplot(x=y_test, y=y_prediction)
plt.title('Sales Prediction')
plt.show()

# Find the error by finding the difference between target testing dataset and the
prediction
y_error = y_test - y_prediction

# Convert the resulting values to dataframe and transpose the index and
columns, rename the columns
```

```
error_data=pd.DataFrame(np.array([y_test,y_prediction,y_error])).T

error_data=error_data.rename(columns={0:'Actual',1:'Predicted',2:'Error',3:'Acc
urcy%'})

error_data


# Plot a line plot between the actual and predicted values

plt.style.use('seaborn')

plt.figure(figsize=(10,3))

no_of_data_show = 30

predicted = list(error_data['Predicted'])[0:no_of_data_show]

actual = list(error_data['Actual'])[0:no_of_data_show]


n=len(predicted)

r = np.arange(n)

width = 0.25



plt.plot(r, predicted, label='Predicted')

plt.plot(r + width, actual, label='Actual')


plt.xlabel("\nNo of Datas")

plt.ylabel("Data Range")

plt.title("Data Accuracy")

plt.legend(fontsize=15)

plt.grid(color='b', linestyle='-', linewidth=0.9)

plt.xticks(np.arange(0, no_of_data_show, step=1))

plt.show()
```

Output

# Find out the number of rows and columns in the dataset

df.shape

```
(6435, 8)
```

# See more info about the dataset

df.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   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
```

# The datatype of Date column is object. Convert it to datatime

df['Date'] = pd.to_datetime(df['Date'])

df.info()

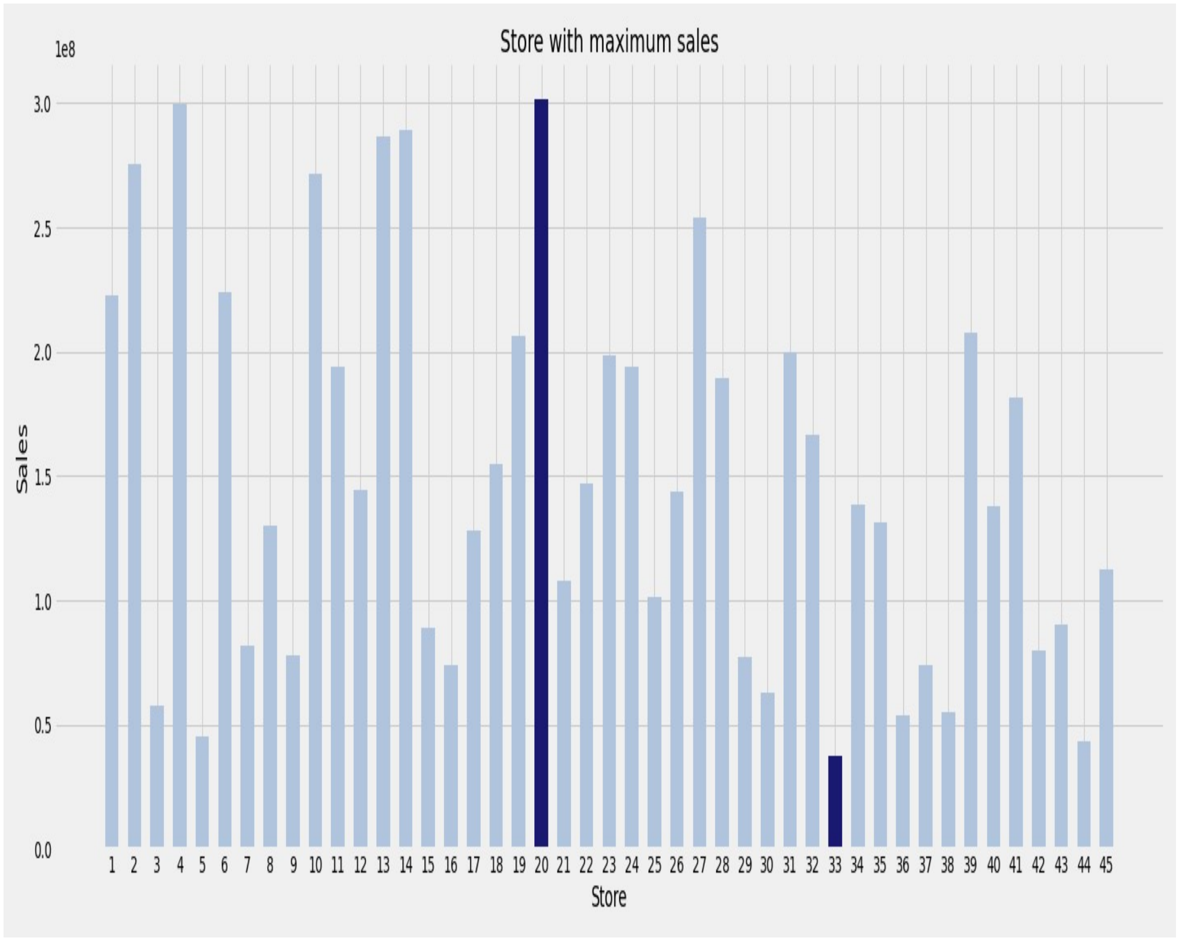
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6435 entries, 0 to 6434
Data columns (total 8 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Store         6435 non-null   int64
 1   Date          6435 non-null   datetime64[ns]
 2   Weekly_Sales  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
```

# Check for null values in the dataset
df.isnull().sum()

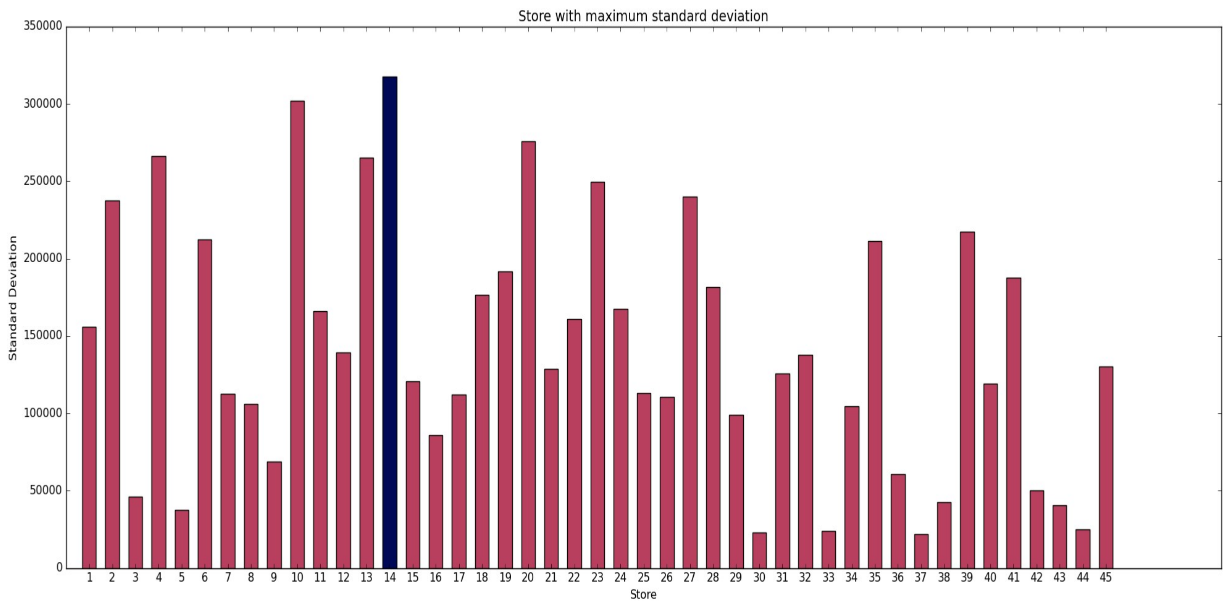
```
Store           0
Date            0
Weekly_Sales    0
Holiday_Flag    0
Temperature     0
Fuel_Price      0
CPI             0
Unemployment    0
dtype: int64
```

Task 1: Which store has the maximum sales

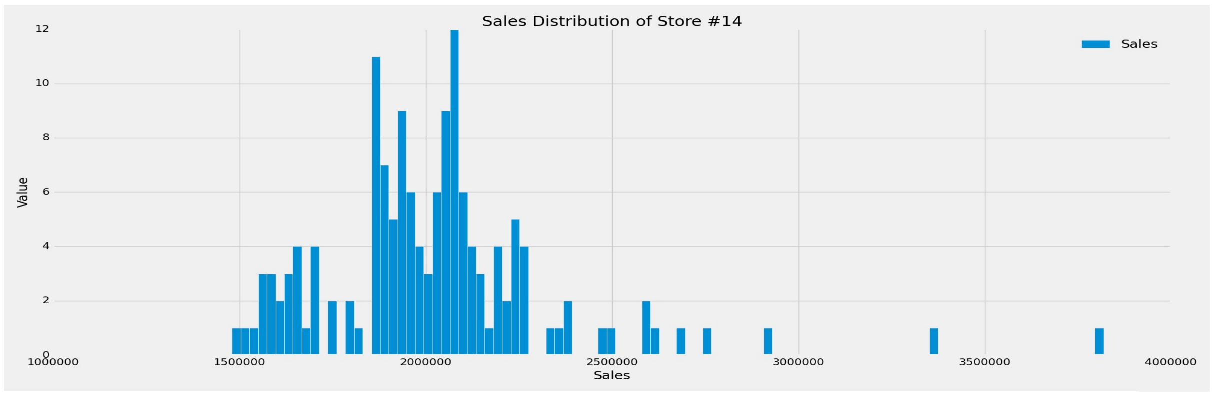


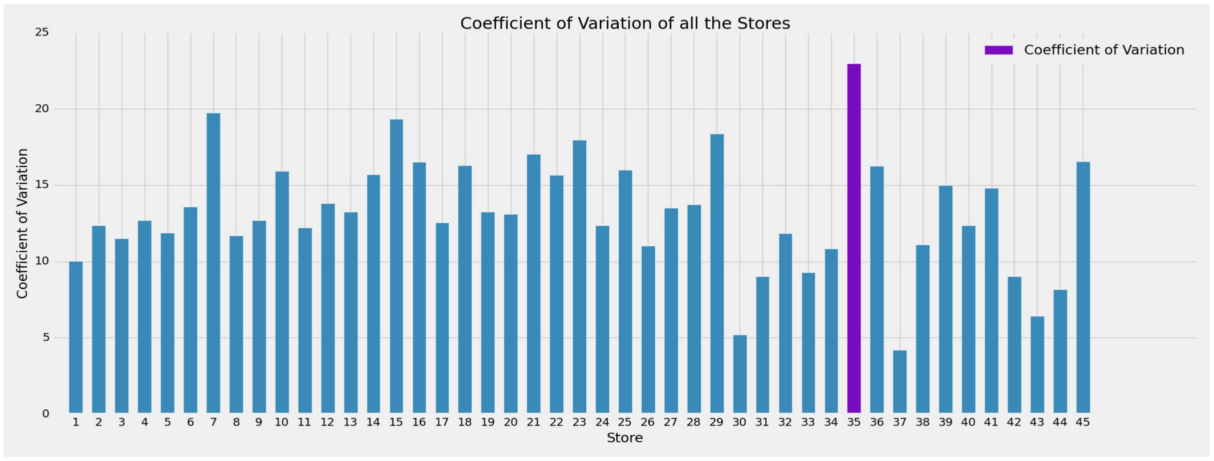
From the above bar graph and statistics, we can conclude that Store #20 has the maximum sales while Store #33 has the minimum sales

Task 2: Which store has maximum standard deviation i.e. the sales vary a l ot. Also, find out the coefficient of mean to standard deviation



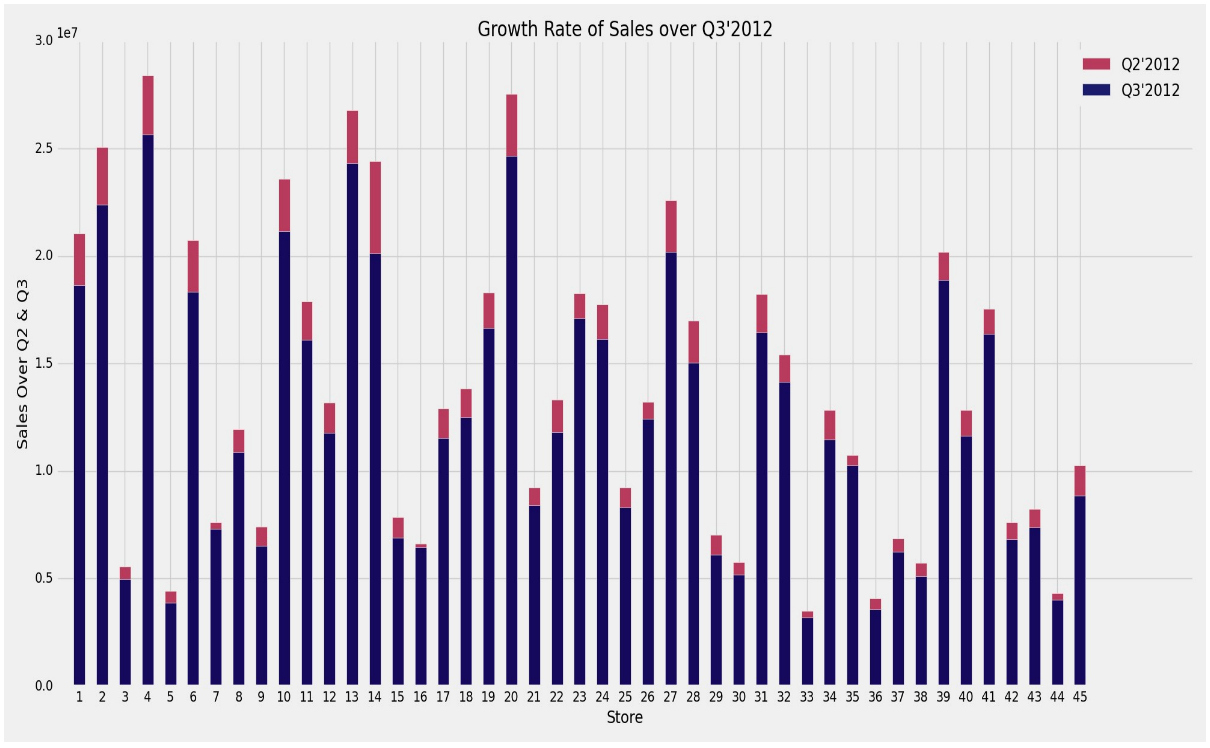
According to the graph, Store #14 has the maximum standard deviation.
Let’s plot the sales of Store #14 to verify.





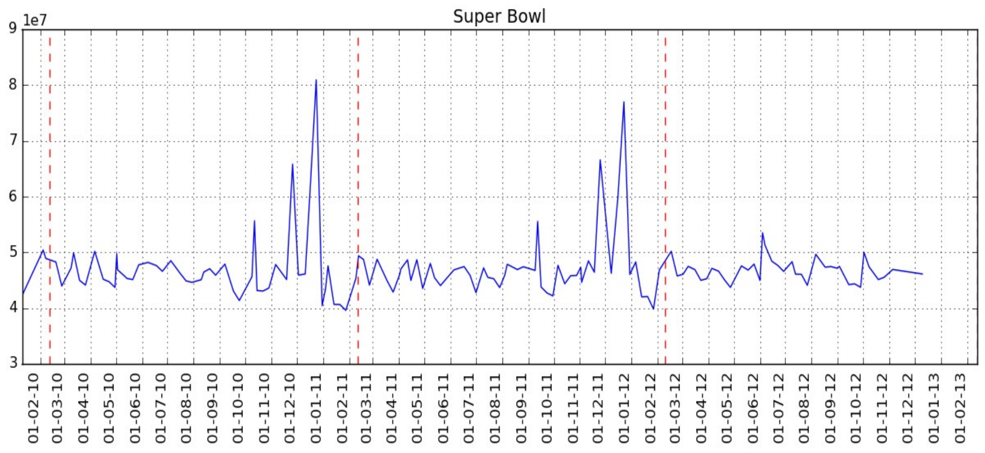
From above, Store #35 has the maximum Coefficient of Variation of all the stores

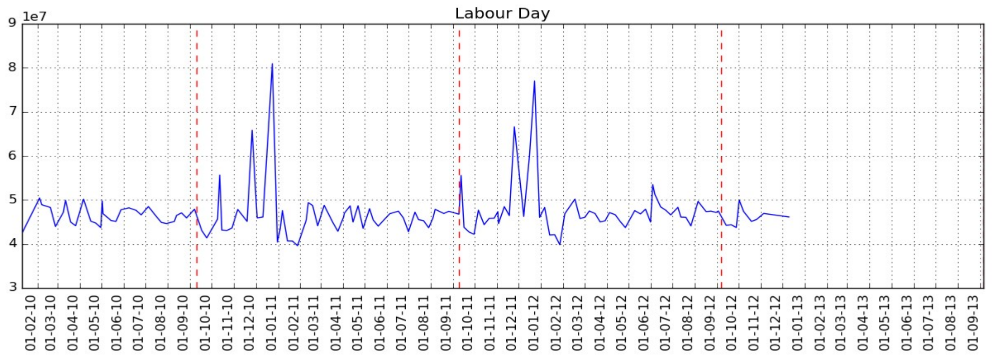
Task 3: Which store/s has good quarterly growth rate in Q3'2012?

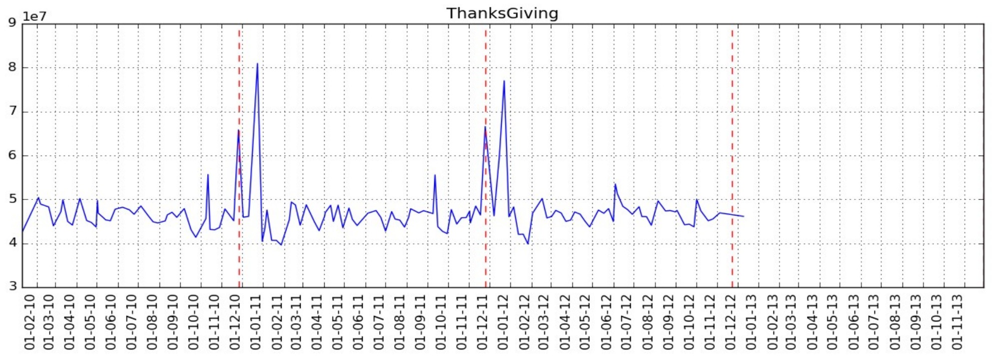


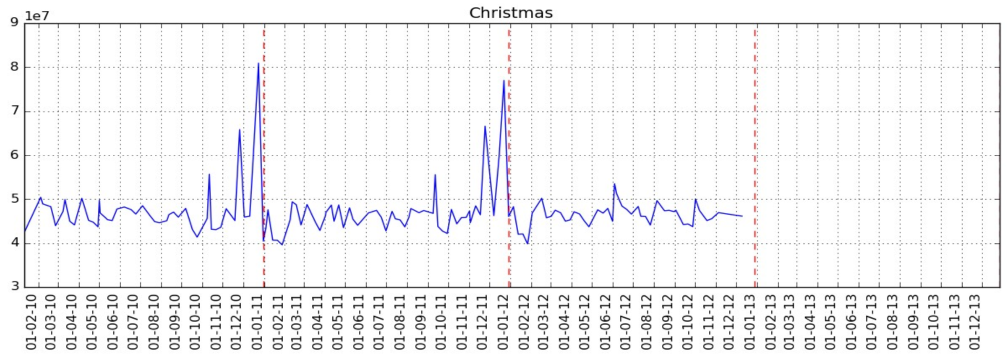
From the above graph, we can clearly see that Store #4 has the maximum growth in sales in Q3'2012

Task 4: Some holidays have a negative impact on sales. Find out holidays which have higher sales than the mean sales in non-holiday season for all stores together

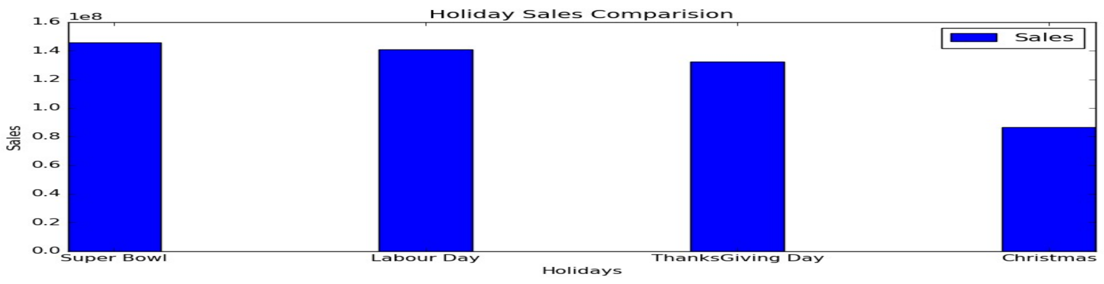




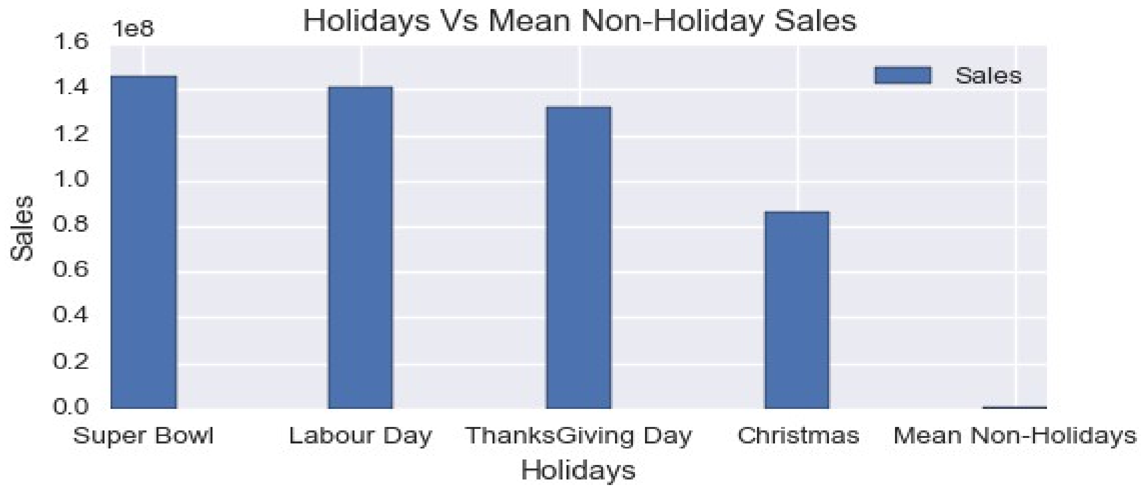




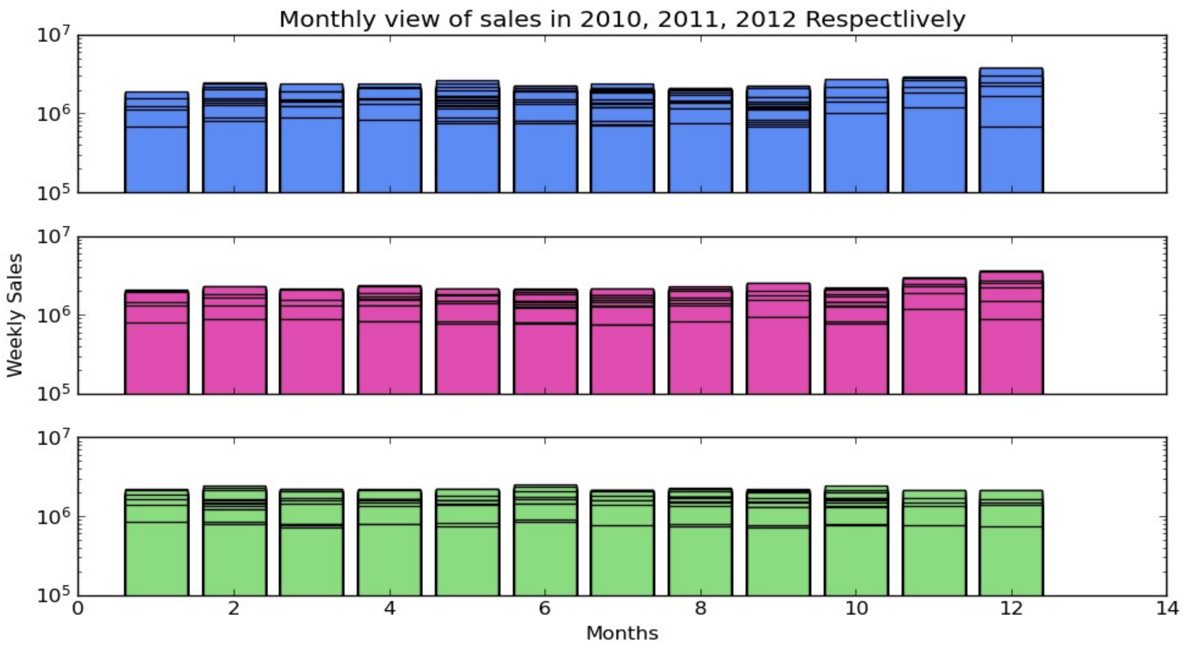
From observing all the four Time-Series charts, we can conclude that compared to the previous month, sales have increased during ThanksGiving Day and sales have decreased during Christmas.



Among all the four holidays, Super Bowl has the most sales, followed by Labour Day and ThanksGiving Day. Christmas has the least sales among the ho lidays.

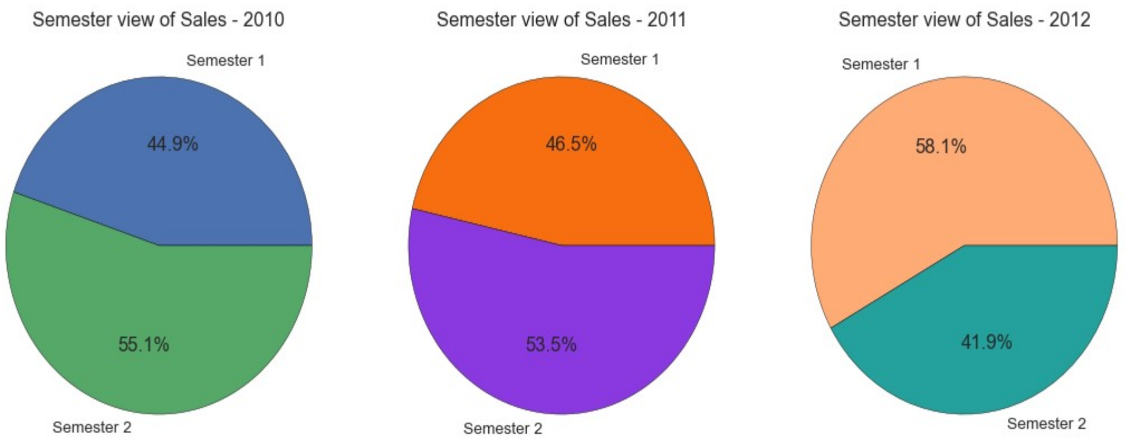


Task 5: Provide a monthly and semester view of sales in units and give insights



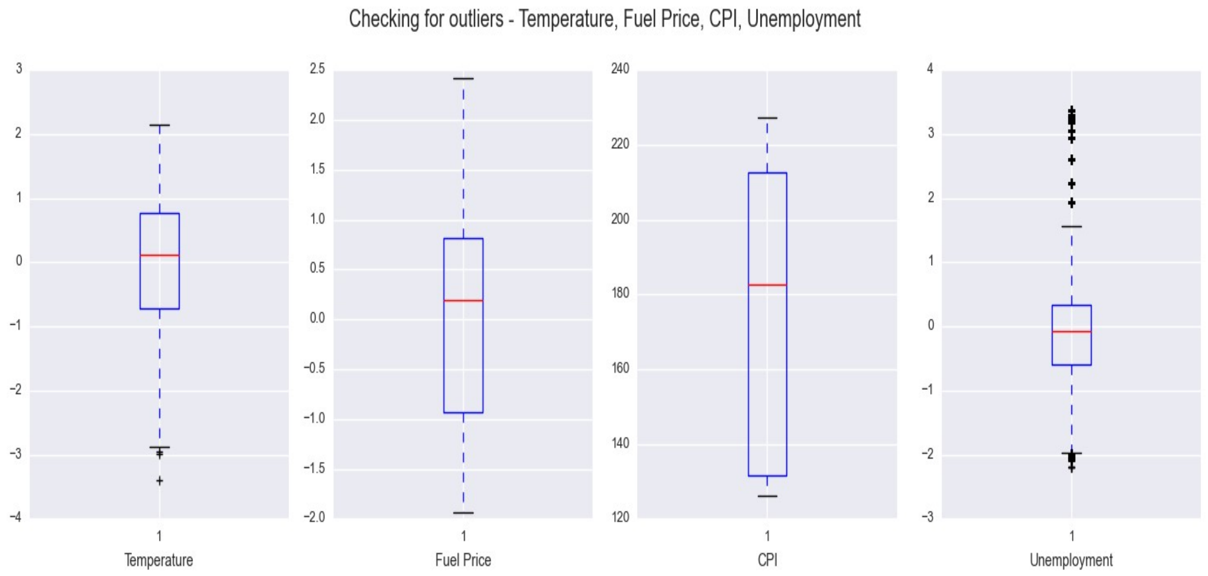
The Holiday months have seen increase in sales compared to its previous months.
The year 2012, when compared to the 2010 and 2011, have seen slight decrease in the sales of the holidays months ThanksGiving Day and Christmas

The Non-Holiday months have similar sales in all the three years, with only slight variation in the year 2010.

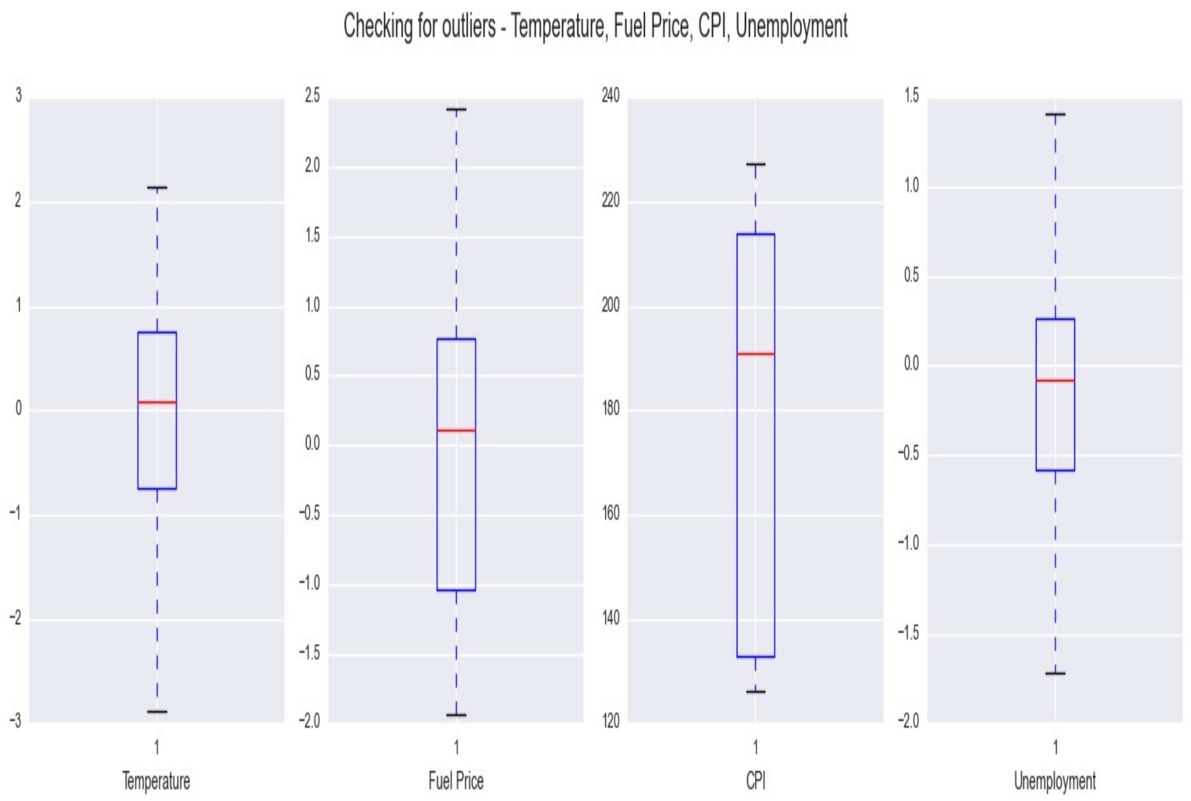


From the above Pie charts, we can see that in the years 2010 & 2011, the second semester, i.e. from July to December has more number of sales when compared to the first semester, i.e. from January to June.
In the year 2012, the situation is reversed, i.e. the first semester has more sales when compared to the second semester.

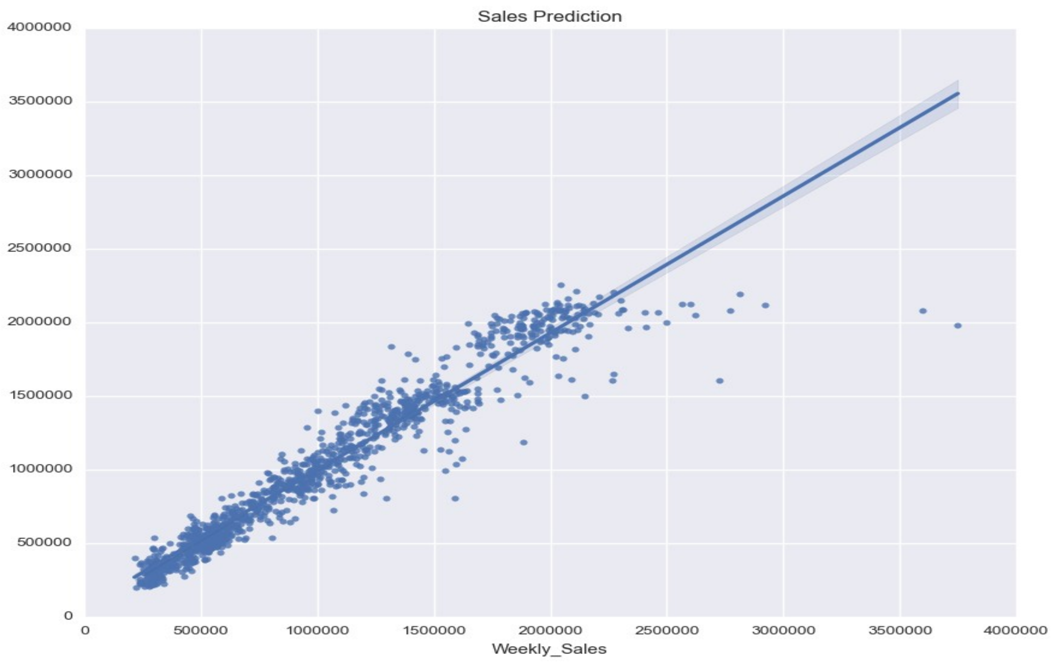
Task 6: Build Prediction model



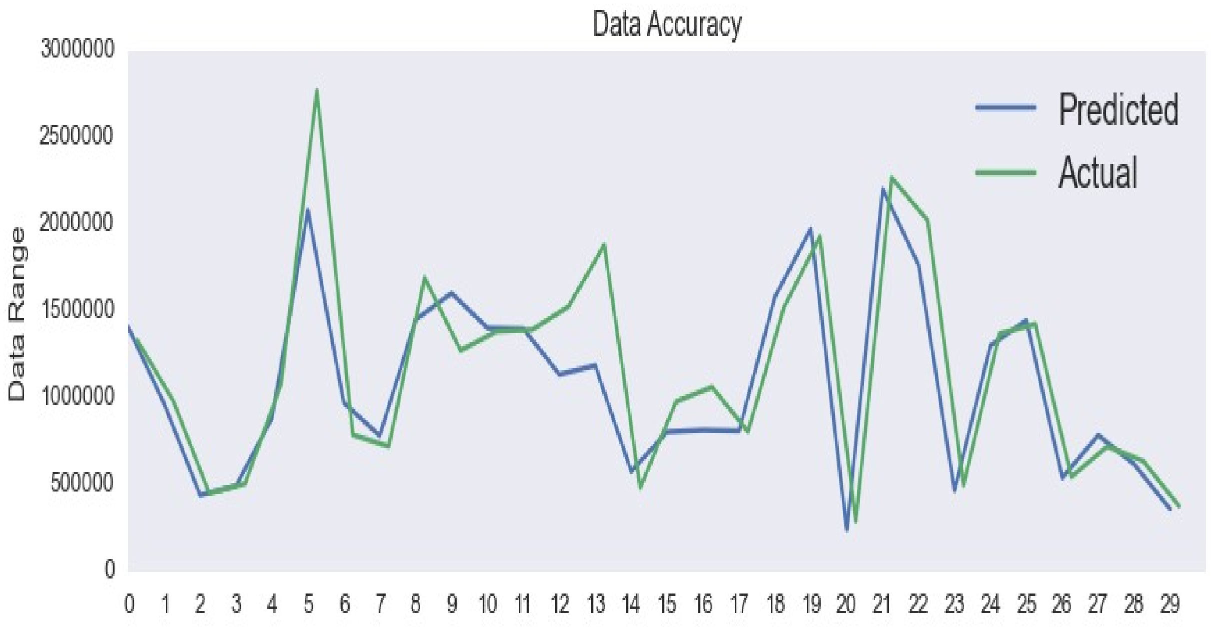
From above, we can see that there are outliers in Unemployment and Temperature fields. Let's remove outliers and check again.



Outliers are removed.



From above, we can predict 92.97% of sales accurately.



From above, we can say that predicted value and actual value are quit close and we can predict up to 92.97% accurately.