Walmart's Weekly Sales Analysis and Prediction

By Buhari Shehu (The Datavestigator)

Table of Contents

1. Introduction

Sales analysis and forecasting are essential tools for businesses to understand and improve their sales performance and make informed decisions about their future sales goals (B2B International, 2018). By analyzing past sales data, businesses can identify trends and patterns that can help them understand what is driving their sales and where they may need to focus their efforts to improve (Small Business Administration, 2021). Forecasting allows businesses to project future sales based on these trends and patterns, helping them to set realistic goals and allocate resources appropriately (Business News Daily, 2021).

The importance of sales analysis and forecasting extends beyond just understanding sales performance. It is also crucial for budgeting and financial planning. By understanding their expected sales, businesses can better plan for expenses and allocate resources to meet their financial goals (Small Business Administration, 2021).

Additionally, sales analysis and forecasting can help businesses identify opportunities for growth and new areas for expansion (B2B International, 2018).

In this project, we will analyze Walmart's weekly sales data across 45 different stores to gain insights into their sales performance and identify trends and patterns. We will then use this data to forecast future sales, which can assist Walmart in making informed strategic decisions. This analysis will provide Walmart with valuable information on how to optimize their sales and allocate resources effectively. Additionally, by understanding how sales vary across different stores, Walmart can identify areas for improvement and potential opportunities for growth. Overall, this project will enable Walmart to gain a deeper understanding of their sales performance and make data-driven decisions to drive future success.

1.1 Dataset description

The file `Walmart.csv` was obtained from Kaggle website. It consists of Walmart's weekly sales from 2010-02-05 to 2012-11-01. The file has the following columns:

- `Store` : the store number
- `Date` : the week of sales
- `Weekly_Sales` : sales for the given store
- `Holiday_Flag` : whether the week is a special holiday week 1 – Holiday week 0 – Non-holiday week
- `Temperature` : Temperature on the day of sale
- `Fuel_Price` : Cost of fuel in the region
- `CPI` : Prevailing consumer price index
- `Unemployment` : Prevailing unemployment rate in percentage

Holiday Events include:

- Super Bowl: 12-Feb-10, 11-Feb-11, 10-Feb-12, 8-Feb-13
- Labour Day: 10-Sep-10, 9-Sep-11, 7-Sep-12, 6-Sep-13
- Thanksgiving: 26-Nov-10, 25-Nov-11, 23-Nov-12, 29-Nov-13
- Christmas: 31-Dec-10, 30-Dec-11, 28-Dec-12, 27-Dec-13

1.2 Importing dependencies

The following packages are essential to running this project successfully: `numpy, pandas, matplotlib, seaborn, and sklearn` .

In [2]:

```
# importing data analysis libraries
import numpy as np
import pandas as pd

# importing visualization libraries
import matplotlib.pyplot as plt
%matplotlib inline

import seaborn as sns
# overriding matplotlib
sns.set()

# import datetime
import datetime as dt

# import the preprocessing classes
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.preprocessing import MinMaxScaler

# import train/test split module
from sklearn.model_selection import train_test_split

# import the regressors
from sklearn.linear_model import LinearRegression
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

# import metrics
from sklearn.metrics import mean_squared_error

# import warnings
import warnings
warnings.filterwarnings('ignore')
```

1.3 Loading the dataset

In [4]:

```
# load the dataset
sales = pd.read_csv('walmart.csv')
sales.head(3)
```

Out[4]:

	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	1	19-02-2010	1611968.17	0	39.93	2.514	211.289143	8.106

2. Data Wrangling

In this section, we will identify and address any errors, inconsistencies, missing values, or duplicate entries in the dataset. This will ensure that the dataset is accurate, consistent, and complete, and will make it more suitable for analysis.

Thus we will address the following questions to ensure the quality and reliability of the dataset:

1. Are there any missing values in the dataset, and if so, what is their extent and data type?
2. Are there any duplicate entries in the dataset?
3. Are there any outliers in the dataset that may impact the analysis?
4. Does the dataset require any feature engineering to better support the analysis goals?

1. Are there any missing values in the dataset, and if so, what is their extent and data type?

In [5]:

```
# get the concise summary of the dataset
```

```
sales.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
```

We can see that the dataset contains 6435 rows and 8 columns. All of the columns are in the appropriate numeric data types, with the exception of the `'Date'` column, which needs to be converted to a `datetime` type. In addition, the feature names will be converted to lowercase for consistency. This information helps us understand the structure and content of the dataset, and identify any necessary data type conversions or formatting changes.

- ***Convert `Date` column data type to `date-time`***

In [6]:

```
# convert Date column data type to date-time
sales['Date'] = pd.to_datetime(sales.Date)
```

In [7]:

```
# check
sales.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
```

The `Date` has been converted to `date-time` data type.

- ***Convert column names to lower case***

In [8]:

```
# convert column names to lower case
sales.columns = [col.lower() for col in sales.columns ]
```

In [9]:

```
# check
sales.columns
```

Out[9]:

```
Index(['store', 'date', 'weekly_sales', 'holiday_flag', 'temperature',
       'fuel_price', 'cpi', 'unemployment'],
```

```
        dtype='object')
```

All column names are now in lower case.

2. Are there any duplicate entries in the dataset?

- ***Check duplicate entries***

In [10]:
```
# check duplicates
sales[sales.duplicated()]
```

Out[10]:

store	date	weekly_sales	holiday_flag	temperature	fuel_price	cpi	unemployment

The dataset is free from duplicate entires.

3. Are there any outliers in the dataset that may impact the analysis?

Outliers, or data points that are significantly different from the rest of the data, can affect the accuracy and reliability of statistical measures such as the mean and standard deviation. To ensure that these measures accurately represent the data, it is necessary to identify and properly handle outliers. To address this issue, we will develop two functions: one to detect outliers and another to count them. These functions will help us identify and understand the impact of outliers on our data, and allow us to make informed decisions about how to handle them.

- ***Create `find_outlier_rows` function***

In [1]:
```
def find_outlier_rows(df, col, level='both'):
    """
    Finds the rows with outliers in a given column of a dataframe.

    This function takes a dataframe and a column as input, and returns the rows
    with outliers in the given column. Outliers are identified using the
    interquartile range (IQR) formula. The optional level parameter allows the
    caller to specify the level of outliers to return, i.e., lower, upper, or both.

    Args:
        df: The input dataframe.
        col: The name of the column to search for outliers.
        level: The level of outliers to return, i.e., 'lower', 'upper', or 'both'.
               Defaults to 'both'.

    Returns:
        A dataframe containing the rows with outliers in the given column.
    """
    # compute the interquartile range
    iqr = df[col].quantile(0.75) - df[col].quantile(0.25)

    # compute the upper and lower bounds for identifying outliers
    lower_bound = df[col].quantile(0.25) - 1.5 * iqr
    upper_bound = df[col].quantile(0.75) + 1.5 * iqr

    # filter the rows based on the level of outliers to return
    if level == 'lower':
        return df[df[col] < lower_bound]
    elif level == 'upper':
        return df[df[col] > upper_bound]
    else:
        return df[(df[col] > upper_bound) | (df[col] < lower_bound)]
```

- ***Create `count_outliers` function***

In [ ]:
```
def count_outliers(df):
    """
    This function takes in a DataFrame and returns a DataFrame containing the count and
    percentage of outliers in each numeric column of the original DataFrame.

    Input:
        df: a Pandas DataFrame containing numeric columns

    Output:
        a Pandas DataFrame containing two columns:
        'outlier_counts': the number of outliers in each numeric column
        'outlier_percent': the percentage of outliers in each numeric column
    """
    # 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, col)) != (

    # 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 the outliers in the columns of the `sales` dataframe***

In [13]:
```
# count the outliers in sales dataframe
count_outliers(sales).sort_values('outlier_counts', ascending=False)
```

Out[13]:

	outlier_counts	outlier_percent
unemployment	481.0	7.47
holiday_flag	450.0	6.99
weekly_sales	34.0	0.53
temperature	3.0	0.05

The above dataframe shows that **`weekly_sales, holiday_flag, temperature and unemployment`** columns all have outliers with **`unemployment`** having the largest outlier percentage, 7%. Let's examine the outliers in each column to decide on how to handle them.

*Examine the **`unemployment`** outliers*

In [14]:
```
# view the summary statistics of unemployment rate
find_outlier_rows(sales, 'unemployment')['unemployment'].describe()
```

Out[14]:
```
count    481.000000
mean      11.447480
std        3.891387
min        3.879000
25%       11.627000
50%       13.503000
75%       14.021000
max       14.313000
Name: unemployment, dtype: float64
```

The minimum and maximum values of these outliers are 3.89% and 14.31% respectively. Majority, greater or equal to 75%, are more than or equal to 11.6% These values are obtainable in reality and will be left intact for the analysis. Thus, median, which rubust to outliers, will be used to measure the centre of the unemployment rate distribution.

Examine the ***holiday_flag*** *outliers*

In [15]:
```
find_outlier_rows(sales, 'holiday_flag')['holiday_flag'].describe()
```

Out[15]:
```
count    450.0
mean       1.0
std        0.0
min        1.0
25%        1.0
50%        1.0
75%        1.0
max        1.0
Name: holiday_flag, dtype: float64
```

We can see that all special holiday weeks form the outliers. This is as a result of the fact that most of the weeks, 93%, are non-special holiday weeks. This will also be left intact.

Examine the ***weekly_sales*** *outliers*

In [16]:
```
# find the outliers in weekly sales
find_outlier_rows(sales, 'weekly_sales')
```

Out[16]:

	store	date	weekly_sales	holiday_flag	temperature	fuel_price	cpi	unemployment
189	2	2010-12-24	3436007.68	0	49.97	2.886	211.064660	8.163
241	2	2011-12-23	3224369.80	0	46.66	3.112	218.999550	7.441
471	4	2010-11-26	2789469.45	1	48.08	2.752	126.669267	7.127
474	4	2010-12-17	2740057.14	0	46.57	2.884	126.879484	7.127
475	4	2010-12-24	3526713.39	0	43.21	2.887	126.983581	7.127
523	4	2011-11-25	3004702.33	1	47.96	3.225	129.836400	5.143
526	4	2011-12-16	2771397.17	0	36.44	3.149	129.898065	5.143
527	4	2011-12-23	3676388.98	0	35.92	3.103	129.984548	5.143
761	6	2010-12-24	2727575.18	0	55.07	2.886	212.916508	7.007
1329	10	2010-11-26	2939946.38	1	55.33	3.162	126.669267	9.003
1332	10	2010-12-17	2811646.85	0	59.15	3.125	126.879484	9.003

	store	date	weekly_sales	holiday_flag	temperature	fuel_price	cpi	unemployment
1333	10	2010-12-24	3749057.69	0	57.06	3.236	126.983581	9.003
1381	10	2011-11-25	2950198.64	1	60.68	3.760	129.836400	7.874
1385	10	2011-12-23	3487986.89	0	48.36	3.541	129.984548	7.874
1758	13	2010-11-26	2766400.05	1	28.22	2.830	126.669267	7.795
1761	13	2010-12-17	2771646.81	0	35.21	2.842	126.879484	7.795
1762	13	2010-12-24	3595903.20	0	34.90	2.846	126.983581	7.795
1810	13	2011-11-25	2864170.61	1	38.89	3.445	129.836400	6.392
1813	13	2011-12-16	2760346.71	0	27.85	3.282	129.898065	6.392
1814	13	2011-12-23	3556766.03	0	24.76	3.186	129.984548	6.392
1901	14	2010-11-26	2921709.71	1	46.15	3.039	182.783277	8.724
1904	14	2010-12-17	2762861.41	0	30.51	3.140	182.517732	8.724
1905	14	2010-12-24	3818686.45	0	30.59	3.141	182.544590	8.724
1957	14	2011-12-23	3369068.99	0	42.27	3.389	188.929975	8.523
2759	20	2010-11-26	2811634.04	1	46.66	3.039	204.962100	7.484
2761	20	2010-10-12	2752122.08	0	24.27	3.109	204.687738	7.484
2762	20	2010-12-17	2819193.17	0	24.07	3.140	204.632119	7.484
2763	20	2010-12-24	3766687.43	0	25.17	3.141	204.637673	7.484
2811	20	2011-11-25	2906233.25	1	46.38	3.492	211.412076	7.082
2814	20	2011-12-16	2762816.65	0	37.16	3.413	212.068504	7.082
2815	20	2011-12-23	3555371.03	0	40.19	3.389	212.236040	7.082
3192	23	2010-12-24	2734277.10	0	22.96	3.150	132.747742	5.287
3764	27	2010-12-24	3078162.08	0	31.34	3.309	136.597273	8.021
3816	27	2011-12-23	2739019.75	0	41.59	3.587	140.528765	7.906

We can note that all the weekly_sales outliers occur either in November or December with one outlier of the outliers that occur in October.

4. Does the dataset require any feature engineering to better support the analysis goals?

Employment rate may be correlated with weekly sales. This will be created from the `unemployment` rate. Also the `date` column will be split into three so that we can analyse the data by year, month or day.

In [17]:

```
# calculate the employment rate
sales['employment'] = 100 - sales['unemployment']

# split the date column
sales['year']= sales['date'].dt.year
sales['month'] = sales['date'].dt.month
sales['day'] = sales['date'].dt.day
sales.head(3)
```

Out[17]:

	store	date	weekly_sales	holiday_flag	temperature	fuel_price	cpi	unemployment	employment	year

	store	date	weekly_sales	holiday_flag	temperature	fuel_price	cpi	unemployment	employment	year
0	1	2010-05-02	1643690.90	0	42.31	2.572	211.096358	8.106	91.894	2010
1	1	2010-12-02	1641957.44	1	38.51	2.548	211.242170	8.106	91.894	2010
2	1	2010-02-19	1611968.17	0	39.93	2.514	211.289143	8.106	91.894	2010

3. Exploratory Data Analysis

This section explores the dataset in order to extract useful information.

1. What are the summary statistics of the dataset?

In [18]:

```
# get the summary statisitcs of the datset
sales.describe()
```

Out[18]:

	store	weekly_sales	holiday_flag	temperature	fuel_price	cpi	unemployment	employmen
count	6435.000000	6.435000e+03	6435.000000	6435.000000	6435.000000	6435.000000	6435.000000	6435.00000(
mean	23.000000	1.046965e+06	0.069930	60.663782	3.358607	171.578394	7.999151	92.00084!
std	12.988182	5.643666e+05	0.255049	18.444933	0.459020	39.356712	1.875885	1.87588!
min	1.000000	2.099862e+05	0.000000	-2.060000	2.472000	126.064000	3.879000	85.68700(
25%	12.000000	5.533501e+05	0.000000	47.460000	2.933000	131.735000	6.891000	91.37800(
50%	23.000000	9.607460e+05	0.000000	62.670000	3.445000	182.616521	7.874000	92.12600(
75%	34.000000	1.420159e+06	0.000000	74.940000	3.735000	212.743293	8.622000	93.10900(
max	45.000000	3.818686e+06	1.000000	100.140000	4.468000	227.232807	14.313000	96.12100(

The weekly transactions occured over the period of three-year (2010-2012) in 45 stores. The maximum weekly sales is $3.8 million and the hottest day was 100°F.

2. What is the distribution of the features of the dataset?

In [19]:

```
# histograms
sales.hist(figsize=(30,20));
```

From the above histograms, we can understand that:

- the number of transactions occurred almost evenly across various stores and years.
- The distribution of `weekly_sales` right-skewed. Only a few of the weekly sales are above 2 million USD.
- The distribution of `temperature` is approximately normal.
- The distribution of `fuel_price` is bi-modal.
- `CPI` formed two clusters.
- `unemployment` rate is near normally distributed.
- Four consecutive months November-February recorded the highest sales.

3. What is the overall trend in sales over time?

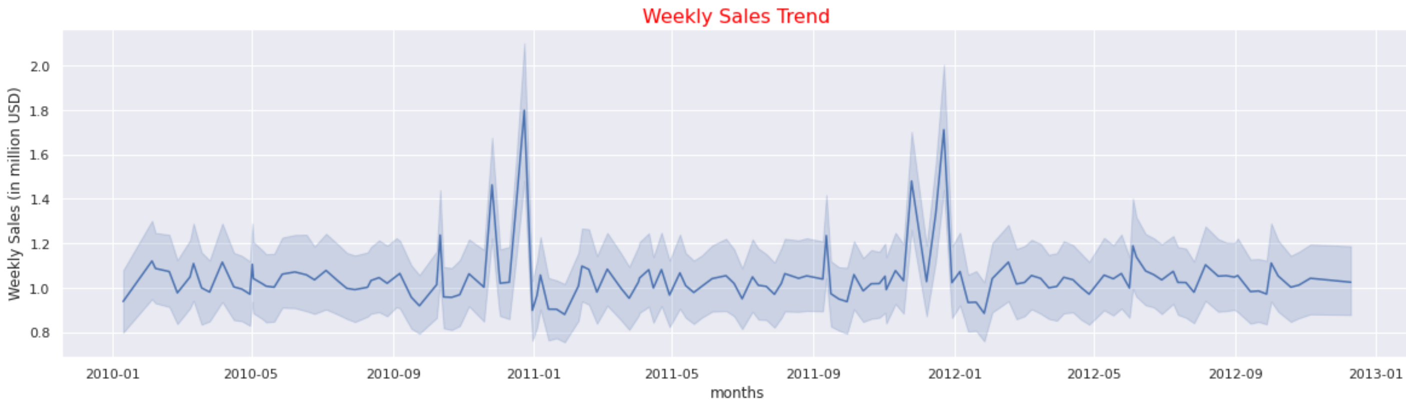
Sales trend analysis involves examining the historical sales data of a business or product over time to understand patterns, trends, and changes in sales performance. It is an important tool for businesses to identify opportunities for growth, understand their customers' behaviour, optimise resources, and make informed decisions about future sales.

We will aggregate the average weekly sales by months for the three year and visualise the trend using a line plot.

In [20]:

```
# plot the line chart of the weekly_sales
fig, ax = plt.subplots(figsize=(20, 5))
sns.lineplot(x=sales.date, y=(sales.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()
```



The line plot reveals that weekly sales at Walmart generally remain stable throughout the year, with the exception of November and December, which experience a significant increase in sales. This trend is likely due to the holiday season, when consumers typically make more purchases and retailers offer promotions and discounts. To capitalize on this behavior, Walmart could consider offering seasonal discounts and promotions, as well as ensuring a seamless and enjoyable shopping experience through their mobile and web outlets during festive periods. By doing so, they can encourage more customers to make purchases and potentially drive up sales.

4. Are there any seasonality trends in the dataset?

Seasonality trends analysis can be extremely valuable for businesses, as it allows us to better forecast future sales, make more informed decisions about inventory and staffing, and understand the drivers of customer demand leading to improved efficiency and profitability.

We will create a pivot table to group the data by month and year and calculate the average sales for each period. We will then plot the average sales of the table using line chart for the three years. This will allow us to see if there are any patterns in the data that repeat at regular intervals.

In [21]:

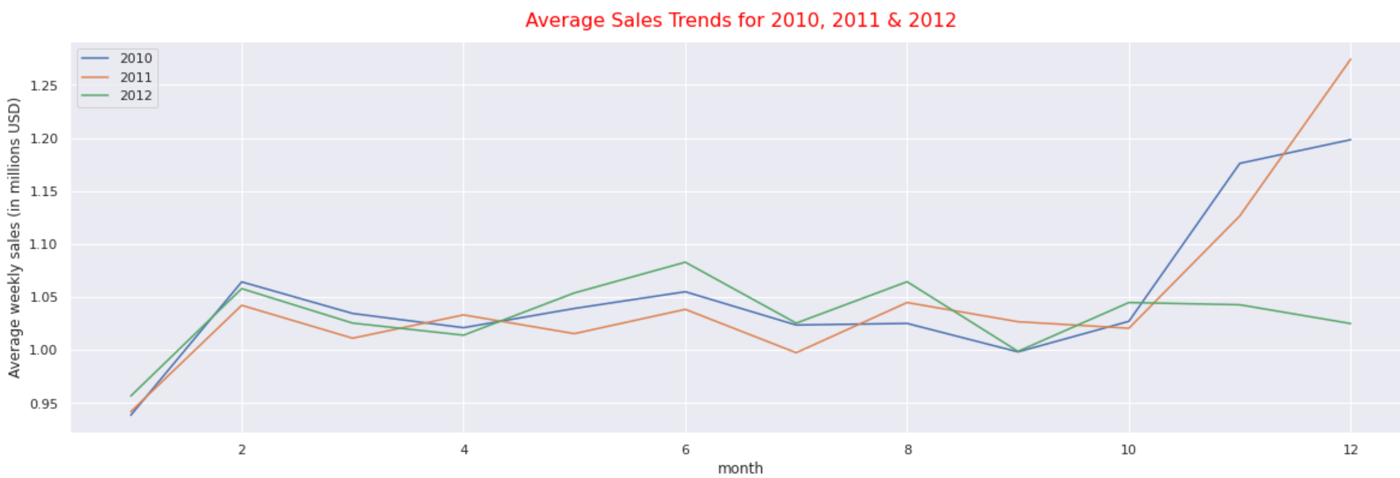
```
# create the pivot table
pivot_table = sales.pivot_table(index='month', columns='year', values='weekly_sales')
# display the pivot table
pivot_table
```

Out[21]:

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

year	2010	2011	2012
month			
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 [22]:
```
# plot the average sales
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':'red',
                                                                    'horizontalalignment':
                                                                    pad=12)

# Add a legend
plt.legend()
plt.show()
```



We can observe that the line charts for the three years for the month of January to October simultaneously follow a sawtooth shape with big rises experienced in November and December due to holidays. This indicates seasonality trends as months do have consistencies in bigger or smaller sales for the three years. We can also observe that although 2011 performed worst than 2010 in terms of average sales for Walmart, the trend was reversed for the year 2012 which performed better than 2010. However, the data for 2012 ends in October, which may explain the significant drop in sales for November."

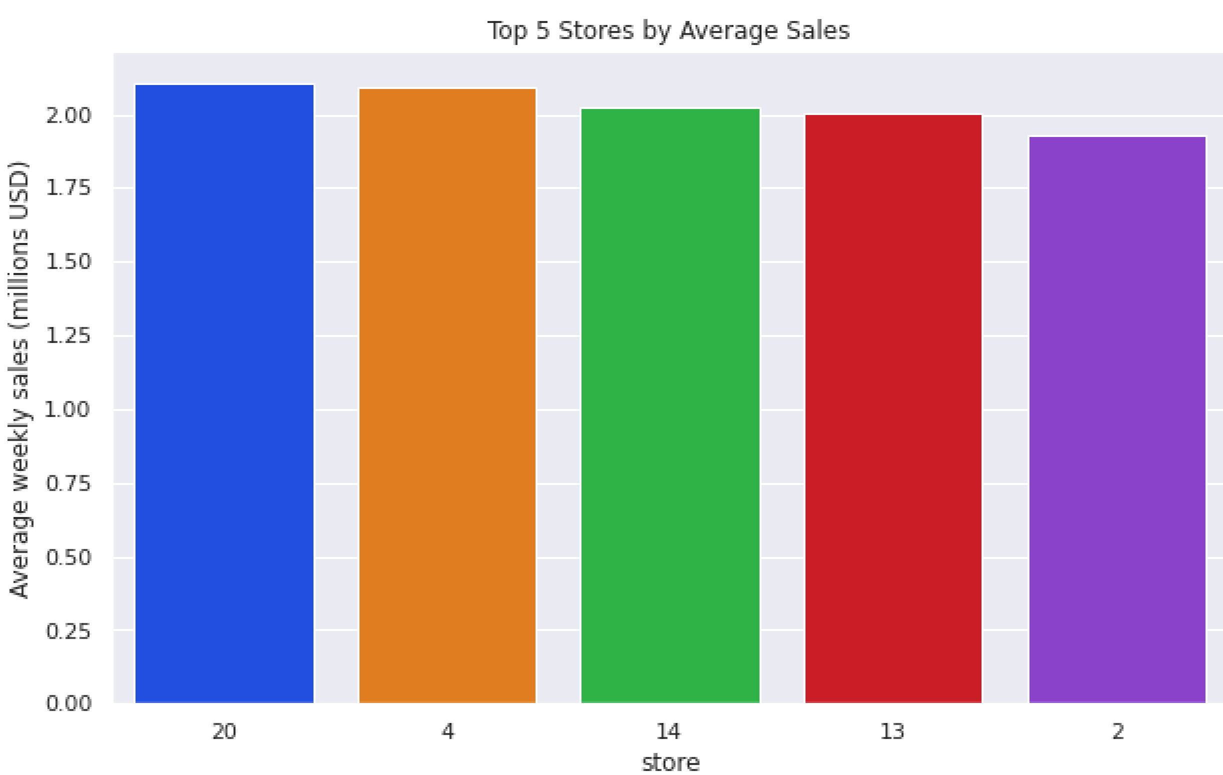
5. Which stores had the highest and lowest average revenues over the years?

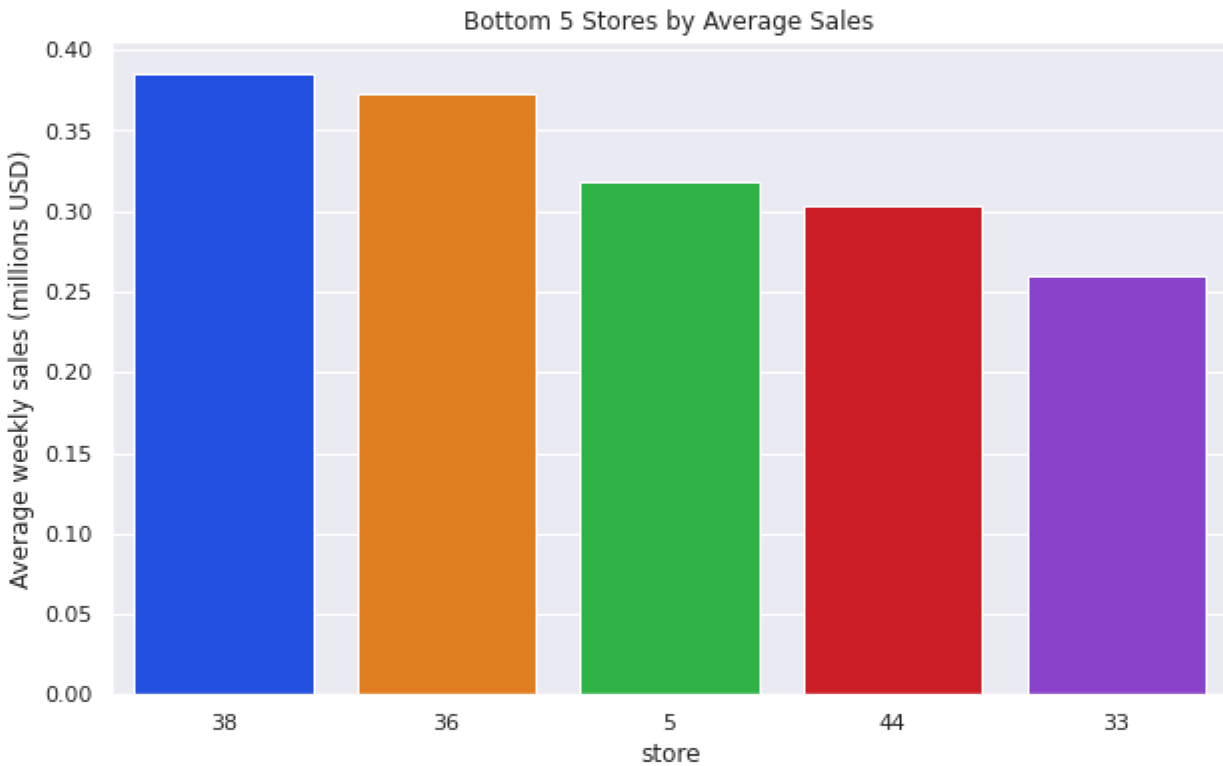
Identifying the top performing and lo performing stores or products in sales analysis can be useful for a variety of purposes. By analysing the sales data for different stores, businesses can identify opportunities for growth, understand customer preferences, optimise inventory levels, and identify potential problems or areas for improvement. Understanding the performance of different stores can inform product development and marketing efforts, as well as help businesses allocate resources more effectively and make more informed business decisions.

We will create a function that takes a dataframe as input and generates two plots showing the top and bottom performing stores in terms of average sales.

In [23]:
```python
def plot_top_and_bottom_stores(df, col):
    """
    Plot the top and bottom 5 stores based on their average weekly sales.

    Parameters:
    df (pandas DataFrame): The dataframe containing the sales data.
    col (str): The name of the column to group the data by.

    Returns:
    None
    """
    # Group the data by the specified column and sort it by sales in descending order
    df = df.groupby(col).mean().sort_values(by='weekly_sales', ascending=False)

    # Select the top 5 and bottom 5 products
    top_stores = df.head(5)
    bottom_stores = df.tail(5)

    # Set the color palette
    sns.set_palette("bright")

    # Create a bar chart of the top 5 products
    fig, ax = plt.subplots(figsize=(10, 6))
    sns.barplot(x=top_stores.index, y=top_stores['weekly_sales']/1e6, order=top_stores.inc
    plt.title('Top 5 Stores by Average Sales')
    plt.ylabel('Average weekly sales (millions USD)')
    plt.show()

    # Create a bar chart of the bottom 5 products
    fig, ax = plt.subplots(figsize=(10, 6))
    sns.barplot(x=bottom_stores.index, y=bottom_stores['weekly_sales']/1e6, order=bottom_s
    plt.title('Bottom 5 Stores by Average Sales')
    plt.ylabel('Average weekly sales (millions USD)')
    plt.show()
```

In [24]:
```python
plot_top_and_bottom_stores(sales, 'store')
```





The graphs show that the top performing stores have relatively stable sales with an average of around $2 million USD. Store 20 appears to be the top performer among these stores, with relatively little variation in sales compared to the other top performers.

On the other hand, the lowest performing stores have higher variations in sales, with the highest sales at around $0.38 million USD. This suggests that there may be more variability in the sales performance of these stores.

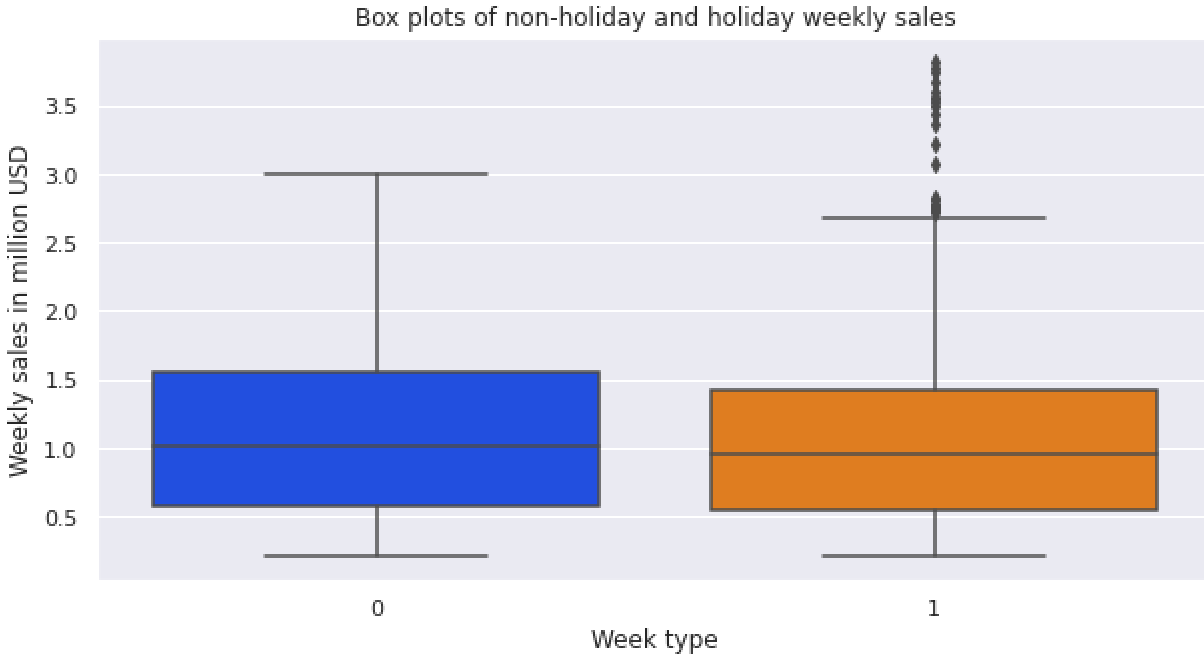
5. How does non-holiday weekly sales compared to holiday weekly sales?

In [25]:

```
# filter out non-holiday and holiday weekly sales
non_holiday_sales = sales[sales['holiday_flag'] == 0]
holiday_sales = sales[sales['holiday_flag'] == 1]
```

In [26]:

```
# plot box plots of non-holiday and holiday weekly sales
fig, ax = plt.subplots(figsize=(10, 5))
sns.boxplot(data=[holiday_sales['weekly_sales']/1e6, non_holiday_sales['weekly_sales']/1e6
plt.ylabel('Weekly sales in million USD')
plt.xlabel('Week type')
plt.title('Box plots of non-holiday and holiday weekly sales')
plt.show()
```

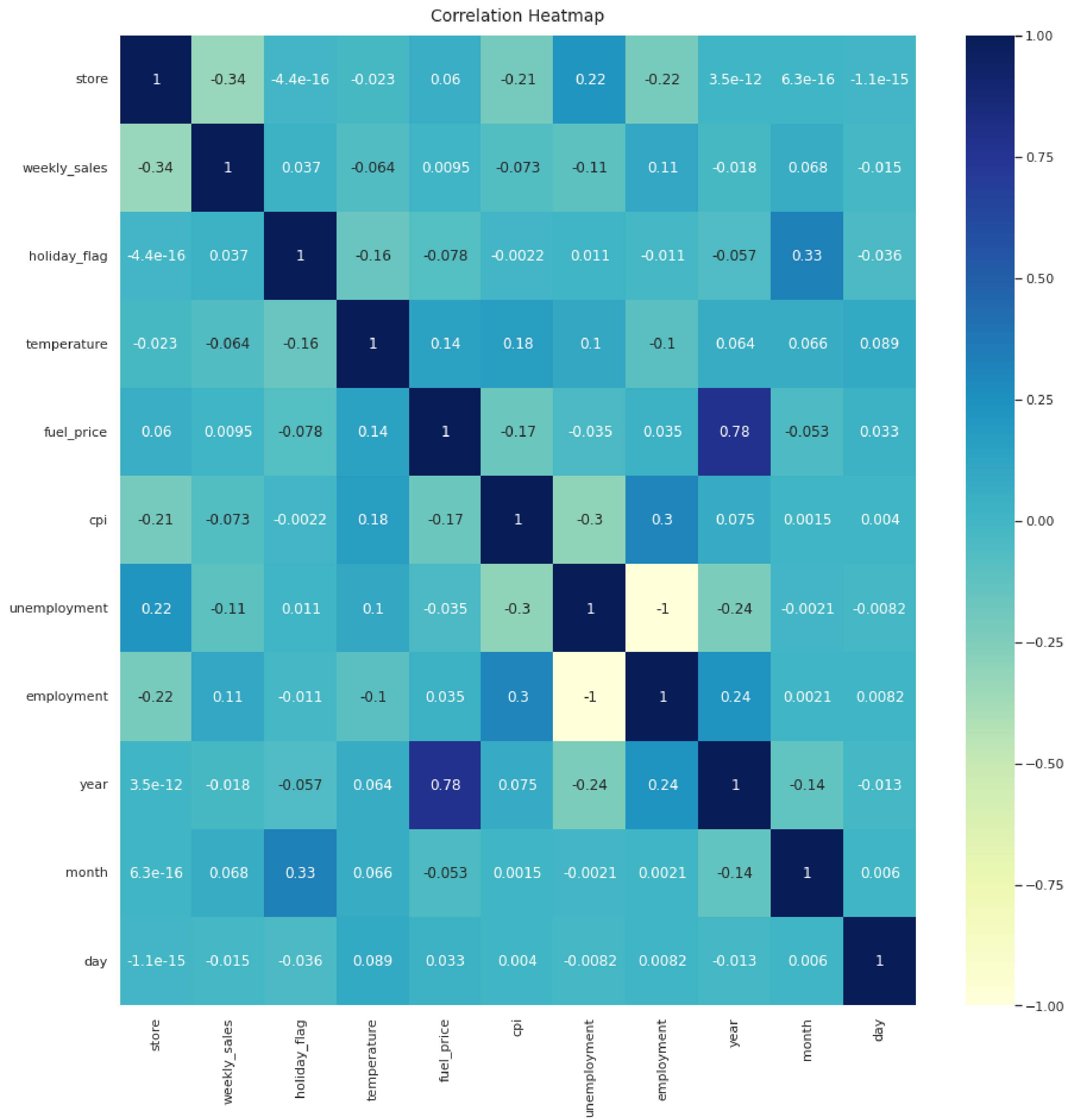


We can see that both holiday and non-holiday weekly sales have similar spread. However, the bigger sales happen during the holiday weeks.

6. Are there correlations between the features of the dataset and weekly_sales?

In [27]:

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



Of all the weaker correlations, employment is the strongest with 0.11 correlation coefficient.

4. Model Selection and Evaluation

In this section, we will evaluate the performance of several different regressors on our data. We will use root mean squared error (RMSE) as our evaluation metric. RMSE is a measure of the difference between the predicted values and the true values. It is calculated as the square root of the mean squared error (MSE), where MSE is the average of the squared differences between the predicted and true values. Lower values of RMSE indicate better performance.

We will fit and evaluate the following regressors:

- Linear Regression

- Decision Tree Regressor
- Random Forest Regressor
- Support Vector Regressor, etc.

We will fit each of these regressors to our training data and make predictions on the test set. Then, we will calculate the RMSE of the predictions and compare the results to choose the best regressor.

To ensure that the original dataset is not modified during the modeling process and to facilitate debugging if needed, we will create a copy of the preprocessed dataset before fitting our various models. This will help to preserve the integrity of the original data and allow us to refer to it if any issues arise during the modeling process.

In [28]:

```
# make a copy of the dataset
sales_copy = sales.copy()
```

In [29]:

```
# drop the date and unemployment columns
sales_copy.drop(['date', 'unemployment'], axis=1, inplace=True)
# check
sales_copy.head()
```

Out[29]:

	store	weekly_sales	holiday_flag	temperature	fuel_price	cpi	employment	year	month	day
0	1	1643690.90	0	42.31	2.572	211.096358	91.894	2010	5	2
1	1	1641957.44	1	38.51	2.548	211.242170	91.894	2010	12	2
2	1	1611968.17	0	39.93	2.514	211.289143	91.894	2010	2	19
3	1	1409727.59	0	46.63	2.561	211.319643	91.894	2010	2	26
4	1	1554806.68	0	46.50	2.625	211.350143	91.894	2010	5	3

In [30]:

```
X = sales_copy.drop('weekly_sales', axis=1)
y = sales_copy['weekly_sales']
```

Scaling the features

Scaling is a preprocessing step that transforms the features of a dataset so that they have a similar scale and can improve the performance of some regression algorithms and facilitate comparison of the model's coefficients. In this project we will use standard scaler to standardize the features of the dataset.

In [31]:

```
# scale the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

Spltting the dataset

To properly evaluate the performance of our dataset and prevent overfitting, we can use cross-validation techniques. One such technique is to split the dataset into a training set and a testing set. The training set is used to train the model, while the testing set is used to evaluate the model's performance. This can help us determine how well the model generalizes to unseen data and can identify any issues with overfitting. It is important to randomly shuffle the data before splitting it into the train and test sets, as this can help ensure that the data is representative of the overall population and not biased in any way.

In [32]:
```
# split the dataset into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_sta
```

Model training and evaluation

In this subsection, we will create a function that will train multiple regressors and compare their performance using the root mean square error (RMSE) metric. We will use the RMSE values to compare the performance of the various regressors and determine which model has the lowest error and is therefore the best fit for our data.

In [ ]:
```
def evaluate_model(model, X_train, y_train, X_test, y_test):
    """
    Evaluate a model on training and test data.

    Parameters
    ----------
    model : object
        A scikit-learn estimator object.
    X_train : array-like or pd.DataFrame
        Training data with shape (n_samples, n_features).
    y_train : array-like
        Training labels with shape (n_samples,).
    X_test : array-like or pd.DataFrame
        Test data with shape (n_samples, n_features).
    y_test : array-like
        Test labels with shape (n_samples,).

    Returns
    -------
    rmse : float
        Root mean squared error between the test labels and the predictions.
    """
    # 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 [34]:
```
def evaluate_regressors_rmses(regressors, regressor_names, X_train, y_train, X_test, y_tes
    """
    This function takes a list of regressors, their names, and the training and test data
    and returns a dataframe with the names of the regressors and their root mean squared e
    on the test data.

    Parameters:
    -----------
    regressors (list): a list of scikit-learn compatible regression models
    regressor_names (list): a list of strings containing the names of the regression model
    X_train (pandas DataFrame): a pandas DataFrame containing the features for training th
    y_train (pandas Series): a pandas Series containing the target values for training the
    X_test (pandas DataFrame): a pandas DataFrame containing the features for testing the
    y_test (pandas Series): a pandas Series containing the target values for testing the r

    Returns:
    --------
    pandas DataFrame: a dataframe containing the names of the regressors and their corresp
    """

    # evaluate the models and compute their RMSE on the test data
```

```python
    rmses = [evaluate_model(regressor, X_train, y_train, X_test, y_test) for regressor in

    # 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 [35]:

```python
# initialize the regressors
linear_regressor = LinearRegression()
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, loss='ls')
knn_regressor = KNeighborsRegressor(n_neighbors=5, weights='uniform')
spline_regressor = make_pipeline(PolynomialFeatures(3), LinearRegression())
```

In [36]:

```python
# collect the list of regressors
regressors = [linear_regressor, polynomial_regressor, ridge_regressor, lasso_regressor, el
              decision_tree_regressor, random_forest_regressor, boosted_tree_regressor, ne
              support_vector_regressor, knn_regressor, spline_regressor]

# collect the names of regressors
regressor_names = ["Linear Regression", "Polynomial Regression", "Ridge Regression", "Lass
                   "Elastic Net Regression", "Decision Tree Regression", "Random Forest Re
                   "Boosted Tree Regression", "Neural Network Regression", "Support Vecto
                   "K-Nearest Neighbour Regression", "Spline Regression"]
```

In [37]:

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

Table of regressors and their RMSEs

Out[37]:

	regressor_name	rmse
0	Random Forest Regression	1.166314e+05
1	Decision Tree Regression	1.574946e+05
2	Boosted Tree Regression	1.750082e+05
3	Spline Regression	4.376419e+05
4	K-Nearest Neighbour Regression	4.606521e+05
5	Polynomial Regression	4.786478e+05

	regressor_name	rmse
6	Ridge Regression	5.209623e+05
7	Lasso Regression	5.209627e+05
8	Linear Regression	5.209628e+05
9	Elastic Net Regression	5.258987e+05
10	Support Vector Regression	5.687996e+05
11	Neural Network Regression	1.185254e+06

Result interpretation

Let's interprete the result by evaluating the rmse value of the best model, the Random Forest Regressor.

In [38]:

```
# evaluate rmse for the regressors
rmse = evaluate_regressors_rmses(regressors, regressor_names, X_train, y_train, X_test, y_
```

In [39]:

```
# pick the best rmse
best_rmse = rmse.iloc[0]['rmse']
# compute the median of the weekly sales
median_sale = sales['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 12.01%
```

The above table shows that Random Forest Regressor outperformed all the regressors with RMSE of 1.17e+05. This provide a good estimate for future sales as it has about 12% average error compared to the typical median sale.

5. Conclusions

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.

Future work

- One area that future studies could explore is the relationship between festive sales and profit margins. By augmenting the dataset with expenses data, it would be possible to investigate whether larger festive sales always translate to larger profit margins. This can inform decisions about marketing and pricing strategies.
- The analysis showed a 500% difference in sales between the top performing and lowest performing stores, which is a significant difference. This suggests that there may be underlying factors contributing to the

performance of these stores. To better understand the reasons behind the performance of these stores, it is necessary to gather additional data and parameters that may be influencing the sales of the top selling products. -Hyperparameter tuning involves adjusting the parameters of a model in order to improve its performance on a given dataset. By iteratively adjusting the parameters of the best model, it is possible to achieve an even better model.

6. References

- Kaggle
- B2B International (2018). Sales Forecasting: The Importance and Benefits.
- Business News Daily (2021). Sales Forecasting: The Importance of Accurate Sales Forecasts.
- Small Business Administration (2021). The Importance of Sales Forecasting.

Email Icon shehubuhari1@gmail.com LinkedIn Icon LinkedIn GitHub Icon GitHub

---

Connecting the facts