pythonwalmartproject

February 19, 2025

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
from matplotlib import dates
from datetime import datetime
import sklearn
import seaborn as sns
```

[2]:
```
#pd.read_csv('Walmart_Store_sales.csv')
#data.head()
```

[3]:
```
data = pd.read_csv("Walmart_Store_sales.csv")
data.head()
```

[4]:
```
data.isna().sum()
```

[5]:
```
data.isnull().sum()
```

[6]:
```
data.shape
```

[7]:
```
data.describe()
```

[8]:
```
data.info()
```

[9]:
```
data.corr()
```

[10]:
```
# Ensure the "Date" column is in datetime format
data['Date'] = pd.to_datetime(data['Date'])

# Extract Day, Month, and Year
data['Day'] = data['Date'].dt.day
data['Month'] = data['Date'].dt.month
data['Year'] = data['Date'].dt.year

# Display the dataframe
data
```

[11]:
```
# 1 ) Which store has maximum sales ?
total_sales= data.groupby('Store')['Weekly_Sales'].sum().sort_values()
total_sales_array = np.array(total_sales)
plt.figure(figsize=(15,7))
plt.xticks(rotation=0)
plt.ticklabel_format(useOffset=False, style='plain', axis='y')
plt.title('Total sales for each store')
plt.xlabel('Store')
plt.ylabel('Total Sales')
total_sales.plot(kind='bar')
```

[12]:
```
# 2. Which store has maximum standard deviation i.e., the sales vary a lot.
#Also, find out the coefficient of mean to standard deviation
data_std = pd.DataFrame(data.groupby('Store')['Weekly_Sales'].std().
 ↪sort_values(ascending=False))
data_std.head(1).index[0] ,data_std.head(1).Weekly_Sales[data_std.head(1).
 ↪index[0]]
```

[13]:
```
# Suppress warnings
import warnings
warnings.filterwarnings('ignore')
# Identify the store with good quarterly growth rate in Q3'2012
# Assuming `data_std` contains a standardized growth rate column for stores
top_store = data_std.head(1).index[0]
# Plot the sales distribution for the identified store
plt.figure(figsize=(15, 7))
sns.displot(data[data['Store'] == top_store]['Weekly_Sales'], kde=True,␣
 ↪bins=30, color='blue')
# Set plot title and labels
plt.title('The Sales Distribution of Store No. ' + str(top_store), fontsize=16)
plt.xlabel('Weekly Sales', fontsize=14)
plt.ylabel('Frequency', fontsize=14)
# Display the plot
plt.show()
```

[14]:
```
#Calculating the coefficient of mean to standard deviation
coef = pd.DataFrame(data.groupby('Store')['Weekly_Sales'].std() /data.
 ↪groupby('Store')['Weekly_Sales'].mean(),
columns=['Coefficient of mean to standard deviation']
                   )
coef_max = coef.sort_values(by='Coefficient of mean to standard␣
 ↪deviation',ascending=False)
coef_max.head(7)
```

[15]:
```
plt.figure(figsize=(15,5))
sns.barplot(x=data.Store, y=data.Weekly_Sales)
```

[16]:
```
# Convert date to datetime format
data['Date'] = pd.to_datetime(data['Date'])
data.info()
```

[17]:
```
# Filter the data for Q2 and Q3 of 2012
quarter_2_sales = data[(data['Date'] >= '2012-04-01') & (data['Date'] <=
↪'2012-06-30')]

quarter_3_sales = data[(data['Date'] >= '2012-07-01') & (data['Date'] <=
↪'2012-09-30')]

# Group by store and sum the weekly sales for Q2 and Q3
quarter_2_sales_grouped = quarter_2_sales.groupby('Store')['Weekly_Sales'].sum()
quarter_3_sales_grouped = quarter_3_sales.groupby('Store')['Weekly_Sales'].sum()

# Calculate the sales difference (Q3 sales - Q2 sales)
sales_diff = quarter_3_sales_grouped - quarter_2_sales_grouped

# Plotting the sales difference between Q2 and Q3 for each store
plt.figure(figsize=(15,7))
sales_diff.plot(kind='bar', color='g', alpha=0.6)

# Adding labels and title
plt.title("Sales Difference between Q3 and Q2 in 2012")
plt.xlabel('Store Number')
plt.ylabel('Sales Difference')
plt.legend(["Q3' 2012 - Q2' 2012"])
plt.show()
```

[18]:
```
# Filter the data for Q2 and Q3 of 2012
quarter_2_sales = data[(data['Date'] >= '2012-04-01') & (data['Date'] <=
↪'2012-06-30')]
quarter_3_sales = data[(data['Date'] >= '2012-07-01') & (data['Date'] <=
↪'2012-09-30')]

# Group by 'Store' and sum 'Weekly_Sales' for Q2 and Q3
quarter_2_sales_grouped = quarter_2_sales.groupby('Store')['Weekly_Sales'].sum()
quarter_3_sales_grouped = quarter_3_sales.groupby('Store')['Weekly_Sales'].sum()

# Calculate the quarterly growth rate
quarterly_growth_rate = ((quarter_3_sales_grouped - quarter_2_sales_grouped) /
↪quarter_2_sales_grouped) * 100

# Sort the growth rate values in descending order and display the top stores
↪with the highest growth rate
quarterly_growth_rate.sort_values(ascending=False).head()
```

[19]:
```
plt.figure(figsize=(15,7))
quarterly_growth_rate.sort_values(ascending=False).plot(kind='bar')
```

[20]:
```
# Some holidays have a negative # 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
Super_Bowl =['12-2-2010', '11-2-2011', '10-2-2012']
Labour_Day = ['10-9-2010', '9-9-2011', '7-9-2012']
Thanksgiving = ['26-11-2010', '25-11-2011', '23-11-2012']
Christmas = ['31-12-2010', '30-12-2011', '28-12-2012']
```

[21]:
```
#Calculating mean sales on holidays :
Super_Bowl_Sales = (pd.DataFrame(data.loc[data.Date.
↪isin(Super_Bowl)]))['Weekly_Sales'].mean()
Labour_Day_Sales = (pd.DataFrame(data.loc[data.Date.
↪isin(Labour_Day)]))['Weekly_Sales'].mean()
Thanksgiving_Sales = (pd.DataFrame(data.loc[data.Date.
↪isin(Thanksgiving)]))['Weekly_Sales'].mean
Christmas_Sales = (pd.DataFrame(data.loc[data.Date.
↪isin(Christmas)]))['Weekly_Sales'].mean()
Super_Bowl_Sales,Labour_Day_Sales,Thanksgiving_Sales,Christmas_Sales
```

[22]:
```
#Calculating mean sales on non-holidays :
Non_Holiday_Sales = data[data['Holiday_Flag'] == 0 ]['Weekly_Sales'].mean()
Non_Holiday_Sales
```

[23]:
```
#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
Mean_Sales = {'Super_Bowl_Sales' : Super_Bowl_Sales,
'Labour_Day_Sales': Labour_Day_Sales,
'Thanksgiving_Sales':Thanksgiving_Sales,
'Christmas_Sales': Christmas_Sales,
'Non_Holiday_Sales': Non_Holiday_Sales}

Mean_Sales
```

[24]:
```
# Provide a monthly and semester view of sales in units and give insights
```

[25]:
```
# Create a figure with a specific size
plt.figure(figsize=(15,7))

# Scatter plot for 2010
plt.scatter(
    data[data['Year'] == 2010]['Month'],
    data[data['Year'] == 2010]['Weekly_Sales'],
```

```python
    label='2010', color='blue', alpha=0.5
)

# Scatter plot for 2011
plt.scatter(
    data[data['Year'] == 2011]['Month'],
    data[data['Year'] == 2011]['Weekly_Sales'],
    label='2011', color='green', alpha=0.5
)

# Scatter plot for 2012
plt.scatter(
    data[data['Year'] == 2012]['Month'],
    data[data['Year'] == 2012]['Weekly_Sales'],
    label='2012', color='red', alpha=0.5
)

# Labeling the axes
plt.xlabel("Months")
plt.ylabel("Weekly Sales")

# Title of the plot
plt.title("Monthly View of Sales (2010 - 2012)")

# Adding a legend to differentiate the years
plt.legend()

# Display the plot
plt.show()
```

[26]:
```python
#Overall Monthly Sales
plt.figure(figsize=(15,7))
plt.bar(data["Month"],data["Weekly_Sales"])
plt.xlabel("Months")
plt.ylabel("Weekly Sales")
plt.title("Monthly view of sales")
plt.show()
```

[27]:
```python
#yearly sales
plt.figure(figsize=(15,7))
data.groupby("Year")[["Weekly_Sales"]].sum().plot(kind='bar',legend=False)
plt.xlabel("Years")
plt.ylabel("Weekly Sales")
plt.title("Yearly view of sales")
plt.show()
```

[28]:
```
# overall monthly sales are higher in the month of December while the yearly␣
↪sales in the year 2011 are the highes
```

[30]:
```
# Suppress warnings
import warnings
warnings.filterwarnings('ignore')

# Dataframe containing the columns of interest
X = data[['Temperature', 'Fuel_Price', 'CPI', 'Unemployment']]

# Set up the subplots
fig, axes = plt.subplots(2, 2, figsize=(16, 16))  # 2 rows, 2 columns for the 4␣
↪variables
axes = axes.flatten()  # Flatten the 2D array of axes into 1D for easy indexing

# Loop through each column and create a boxplot
for i, column in enumerate(X):
    sns.boxplot(x=data[column], ax=axes[i])

plt.tight_layout()  # Adjust layout to prevent overlap
plt.show()
```

[39]:
```
# Dropping outliers
data_clean = data[
    (data['Unemployment'] < 10) & (data['Unemployment'] > 4.5) &
    (data['Temperature'] < 100) & (data['Temperature'] > -10) &
    (data['Fuel_Price'] < 5) & (data['Fuel_Price'] > 1) &
    (data['CPI'] < 250) & (data['CPI'] > 80)
]

# Check the cleaned data
print(data_clean.head())
```

[41]:
```
import warnings

# Suppress warnings
warnings.filterwarnings('ignore')

# Dataframe containing the columns of interest
X = data_clean[['Temperature', 'Fuel_Price', 'CPI', 'Unemployment']]

# Set up the subplots (2 rows, 2 columns for the 4 variables)
fig, axes = plt.subplots(2, 2, figsize=(16, 16))
axes = axes.flatten()  # Flatten the 2D array of axes into 1D for easy indexing

# Loop through each column and create a boxplot
for i, column in enumerate(X):
```

```
    sns.boxplot(x=data_clean[column], ax=axes[i])

# Adjust layout to prevent overlap
plt.tight_layout()
plt.show()
```

[43]:
```
# Linear Regression :
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.linear_model import LinearRegression
X = data_clean[['Store','Fuel_Price','CPI','Unemployment','Day','Month','Year']]
Y = data_clean['Weekly_Sales']
X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=0.2)
```

[45]:
```
import warnings

# Suppress warnings
warnings.filterwarnings('ignore')

# Assuming X_train, Y_train, X_test, and Y_test are already defined
print('Linear Regression:')
print()

# Initialize and fit the model
reg = LinearRegression()
reg.fit(X_train, Y_train)

# Make predictions
Y_pred = reg.predict(X_test)

# Evaluate the model
print('Accuracy:', reg.score(X_train, Y_train) * 100)  # R^2 score
print('Mean Absolute Error:', metrics.mean_absolute_error(Y_test, Y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(Y_test, Y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(Y_test,␣
 ↪Y_pred)))

# Scatter plot of true vs predicted values
sns.scatterplot(x=Y_test, y=Y_pred)
```

[47]:
```
import warnings

# Suppress warnings
warnings.filterwarnings('ignore')

# Initialize the Random Forest Regressor
print('Random Forest Regressor:')
```

```
print()
rfr = RandomForestRegressor()
rfr.fit(X_train, Y_train)

# Make predictions
Y_pred = rfr.predict(X_test)

# Evaluate the model
print('Accuracy:', rfr.score(X_test, Y_test) * 100)  # R^2 score
print('Mean Absolute Error:', metrics.mean_absolute_error(Y_test, Y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(Y_test, Y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(Y_test,
 ↪Y_pred)))

# Scatter plot of true vs predicted values
sns.scatterplot(x=Y_test, y=Y_pred)
```

[ ]:

```

```