```
# Import necessary libraries
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import dates
from datetime import datetime

import warnings
warnings.simplefilter(action="ignore", category=FutureWarning)


# Load dataset
data = pd.read_csv('Walmart_Store_sales.csv')
print(data)

# Convert date to datetime format and show dataset information
data['Date'] =  pd.to_datetime(data['Date'],dayfirst=True)
data.info()

# checking for missing values
data.isnull().sum()

# Splitting Date and create new columns (Day, Month, and Year)
data["Day"]= pd.DatetimeIndex(data['Date']).day
data['Month'] = pd.DatetimeIndex(data['Date']).month
data['Year'] = pd.DatetimeIndex(data['Date']).year
print(data)

# Q1: Which store has minimum and maximum sales?

plt.figure(figsize=(15,7))

# Sum Weekly_Sales for each store, then sortded by total sales
total_sales_for_each_store =
data.groupby('Store')['Weekly_Sales'].sum().sort_values()
total_sales_for_each_store_array = np.array(total_sales_for_each_store) # convert
to array

# Assigning a specific color for the stores have the lowest and highest sales
clrs = ['lightsteelblue' if ((x < max(total_sales_for_each_store_array)) and (x >
min(total_sales_for_each_store_array))) else 'midnightblue' for x in
total_sales_for_each_store_array]
```

```
ax = total_sales_for_each_store.plot(kind='bar',color=clrs);

# store have minimum sales
p = ax.patches[0]
print(type(p.get_height()))
ax.annotate("The store has minimum sales is 33 with {0:.2f}
$".format((p.get_height())), xy=(p.get_x(), p.get_height()), xycoords='data',
            xytext=(0.17, 0.32), textcoords='axes fraction',
            arrowprops=dict(arrowstyle="->", connectionstyle="arc3"),
            horizontalalignment='center', verticalalignment='center')


# store have maximum sales
p = ax.patches[44]
ax.annotate("The store has maximum sales is 20 with {0:.2f}
$".format((p.get_height())), xy=(p.get_x(), p.get_height()), xycoords='data',
            xytext=(0.82, 0.98), textcoords='axes fraction',
            arrowprops=dict(arrowstyle="->", connectionstyle="arc3"),
            horizontalalignment='center', verticalalignment='center')


# plot properties
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');

# Q2: Which store has maximum standard deviation i.e., the sales vary a lot.
Also, find out the coefficient of mean to standard deviation?

# Which store has maximum standard deviation
data_std =
pd.DataFrame(data.groupby('Store')['Weekly_Sales'].std().sort_values(ascending=Fa
lse))
print("The store has maximum standard deviation is
"+str(data_std.head(1).index[0])+" with {0:.0f}
$".format(data_std.head(1).Weekly_Sales[data_std.head(1).index[0]]))

# Distribution of store has maximum standard deviation
plt.figure(figsize=(15,7))
sns.displot(data[data['Store'] == data_std.head(1).index[0]]['Weekly_Sales'])
plt.title('The Sales Distribution of Store #'+ str(data_std.head(1).index[0]));

# Coefficient of mean to standard deviation
```

```
coef_mean_std = pd.DataFrame(data.groupby('Store')['Weekly_Sales'].std() /
data.groupby('Store')['Weekly_Sales'].mean())
coef_mean_std = coef_mean_std.rename(columns={'Weekly_Sales':'Coefficient of mean
to standard deviation'})
print(coef_mean_std)

# Distribution of store has maximum coefficient of mean to standard deviation
coef_mean_std_max = coef_mean_std.sort_values(by='Coefficient of mean to standard
deviation')
plt.figure(figsize=(15,7))
sns.displot(data[data['Store'] ==
coef_mean_std_max.tail(1).index[0]]['Weekly_Sales'])
plt.title('The Sales Distribution of Store
#'+str(coef_mean_std_max.tail(1).index[0]));

# Q3: Which store/s has good quarterly growth rate in Q3'2012
plt.figure(figsize=(15,7))

# Sales for third quarterly in 2012
Q3 = data[(data['Date'] > '2012-07-01') & (data['Date'] < '2012-09-
30')].groupby('Store')['Weekly_Sales'].sum()

# Sales for second quarterly in 2012
Q2 = data[(data['Date'] > '2012-04-01') & (data['Date'] < '2012-06-
30')].groupby('Store')['Weekly_Sales'].sum()

# Plotting the difference between sales for second and third quarterly
Q2.plot(ax=Q3.plot(kind='bar',legend=True),kind='bar',color='r',alpha=0.2,legend=
True);
plt.legend(["Q3' 2012", "Q2' 2012"]);


#  store/s has good quarterly growth rate in Q3'2012 -
.sort_values(by='Weekly_Sales')
print('Store have good quarterly growth rate in Q3'2012 is Store
'+str(Q3.idxmax())+' With '+str(Q3.max())+' $')

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

    for day in holiday_dates:
        day = str(day)
        # print(day)
        # print(datetime.strptime(day, '%Y-%m-%d %H:%M:%S'))
```

```
        day = datetime.strptime(day, '%Y-%m-%d %H:%M:%S')
        plt.axvline(x=day, linestyle='--', c='r')


    plt.title(holiday_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.show()


total_sales = data.groupby('Date')['Weekly_Sales'].sum().reset_index()
Super_Bowl =pd.to_datetime(['12-2-2010', '11-2-2011', '10-2-2012'],dayfirst=True)
Labour_Day =  pd.to_datetime(['10-9-2010', '9-9-2011', '7-9-2012'],dayfirst=True)
Thanksgiving =  pd.to_datetime(['26-11-2010', '25-11-2011', '23-11-
2012'],dayfirst=True)
Christmas = pd.to_datetime(['31-12-2010', '30-12-2011', '28-12-
2012'],dayfirst=True)

plot_line(total_sales,Super_Bowl,'Super Bowl')
plot_line(total_sales,Labour_Day,'Labour Day')
plot_line(total_sales,Thanksgiving,'Thanksgiving')
plot_line(total_sales,Christmas,'Christmas')

data.loc[data.Date.isin(Super_Bowl)]

# Yearly Sales in holidays
Super_Bowl_df =
pd.DataFrame(data.loc[data.Date.isin(Super_Bowl)].groupby('Year')['Weekly_Sales']
.sum())
Thanksgiving_df =
pd.DataFrame(data.loc[data.Date.isin(Thanksgiving)].groupby('Year')['Weekly_Sales
'].sum())
Labour_Day_df =
pd.DataFrame(data.loc[data.Date.isin(Labour_Day)].groupby('Year')['Weekly_Sales']
.sum())
Christmas_df =
pd.DataFrame(data.loc[data.Date.isin(Christmas)].groupby('Year')['Weekly_Sales'].
sum())

Super_Bowl_df.plot(kind='bar',legend=False,title='Yearly Sales in Super Bowl
holiday')
```

```
Thanksgiving_df.plot(kind='bar',legend=False,title='Yearly Sales in Thanksgiving
holiday')
Labour_Day_df.plot(kind='bar',legend=False,title='Yearly Sales in Labour_Day
holiday')
Christmas_df.plot(kind='bar',legend=False,title='Yearly Sales in Christmas
holiday')

# Monthly view of sales for each years
plt.scatter(data[data.Year==2010]["Month"],data[data.Year==2010]["Weekly_Sales"])
plt.xlabel("months")
plt.ylabel("Weekly Sales")
plt.title("Monthly view of sales in 2010")
plt.show()

plt.scatter(data[data.Year==2011]["Month"],data[data.Year==2011]["Weekly_Sales"])
plt.xlabel("months")
plt.ylabel("Weekly Sales")
plt.title("Monthly view of sales in 2011")
plt.show()

plt.scatter(data[data.Year==2012]["Month"],data[data.Year==2012]["Weekly_Sales"])
plt.xlabel("months")
plt.ylabel("Weekly Sales")
plt.title("Monthly view of sales in 2012")
plt.show()


# Monthly view of sales for all years
plt.figure(figsize=(10,6))
plt.bar(data["Month"],data["Weekly_Sales"])
plt.xlabel("months")
plt.ylabel("Weekly Sales")
plt.title("Monthly view of sales")


# Yearly view of sales
plt.figure(figsize=(10,6))
data.groupby("Year")[["Weekly_Sales"]].sum().plot(kind='bar',legend=False)
plt.xlabel("years")
plt.ylabel("Weekly Sales")
plt.title("Yearly view of sales");


# find outliers
fig, axs = plt.subplots(4,figsize=(6,18))
```

```
X = data[['Temperature','Fuel_Price','CPI','Unemployment']]
for i,column in enumerate(X):
    sns.boxplot(data[column], ax=axs[i])


# drop the outliers
data_new = data[(data['Unemployment']<10) & (data['Unemployment']>4.5) &
(data['Temperature']>10)]
print(data_new)



# check outliers
fig, axs = plt.subplots(4,figsize=(6,18))
X = data_new[['Temperature','Fuel_Price','CPI','Unemployment']]
for i,column in enumerate(X):
    sns.boxplot(data_new[column], ax=axs[i])


# Import sklearn
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.linear_model import LinearRegression


# Select features and target
X = data_new[['Store','Fuel_Price','CPI','Unemployment','Day','Month','Year']]
y = data_new['Weekly_Sales']

# Split data to train and test (0.80:0.20)
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2)



# Linear Regression model
print('Linear Regression:')
print()
reg = LinearRegression()
reg.fit(X_train, y_train)
y_pred = reg.predict(X_test)
print('Accuracy:',reg.score(X_train, y_train)*100)


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)))


sns.scatterplot(y_pred, y_test);


# Random Forest Regressor
print('Random Forest Regressor:')
print()
rfr = RandomForestRegressor(n_estimators = 400,max_depth=15,n_jobs=5)
rfr.fit(X_train,y_train)
y_pred=rfr.predict(X_test)
print('Accuracy:',rfr.score(X_test, y_test)*100)

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)))


sns.scatterplot(y_pred, y_test);
```