EDA

Author: Om Patil

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
df = pd.read_csv('Dataset_deeplearning.csv')
```

```
# Converting 'date' to datetime format
df['date'] = pd.to_datetime(df['date'])

df['month'] = df['date'].dt.month
df['day_of_week'] = df['date'].dt.dayofweek  # Monday=0, Sunday=6
df['is_weekend'] = df['day_of_week'].apply(lambda x: 1 if x >= 5 else 0)
```

```
# Display basic info
print(df.info())
```

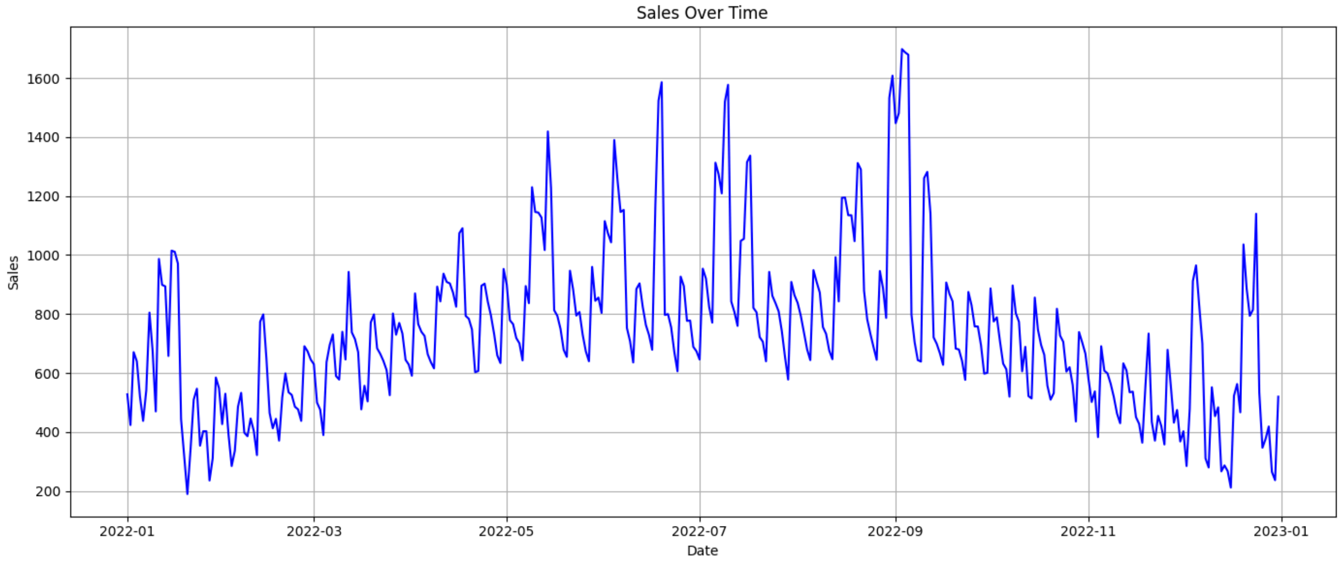
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 365 entries, 0 to 364
Data columns (total 7 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   date         365 non-null    datetime64[ns]
 1   temperature  365 non-null    float64
 2   promotions   365 non-null    int64
 3   sales        365 non-null    int64
 4   month        365 non-null    int32
 5   day_of_week  365 non-null    int32
 6   is_weekend   365 non-null    int64
dtypes: datetime64[ns](1), float64(1), int32(2), int64(3)
memory usage: 17.2 KB
None
```

```
#Display summary stats
print(df.describe())
```

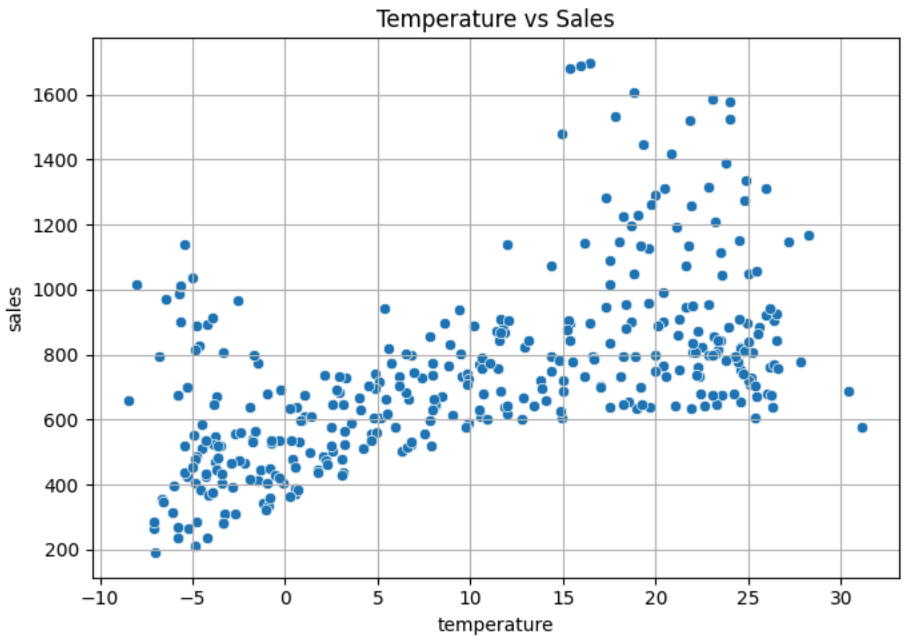
```
                                date  temperature  promotions        sales  \
count                            365   365.000000  365.000000   365.000000
mean   2022-07-01 23:59:59.999999744    10.019890    0.471233   736.972603
min              2022-01-01 00:00:00    -8.450000    0.000000   190.000000
25%              2022-04-02 00:00:00    -0.320000    0.000000   552.000000
50%              2022-07-02 00:00:00     9.820000    0.000000   705.000000
75%              2022-10-01 00:00:00    20.380000    0.000000   864.000000
max              2022-12-31 00:00:00    31.170000    4.000000  1698.000000
std                              NaN    10.890047    0.993036   275.646810

            month  day_of_week  is_weekend
count  365.000000   365.000000  365.000000
mean     6.526027     3.005479    0.287671
min      1.000000     0.000000    0.000000
25%      4.000000     1.000000    0.000000
50%      7.000000     3.000000    0.000000
75%     10.000000     5.000000    1.000000
max     12.000000     6.000000    1.000000
std      3.452584     2.002738    0.453298
```

```
# Correlation matrix
correlation_matrix = df.corr(numeric_only=True)
```

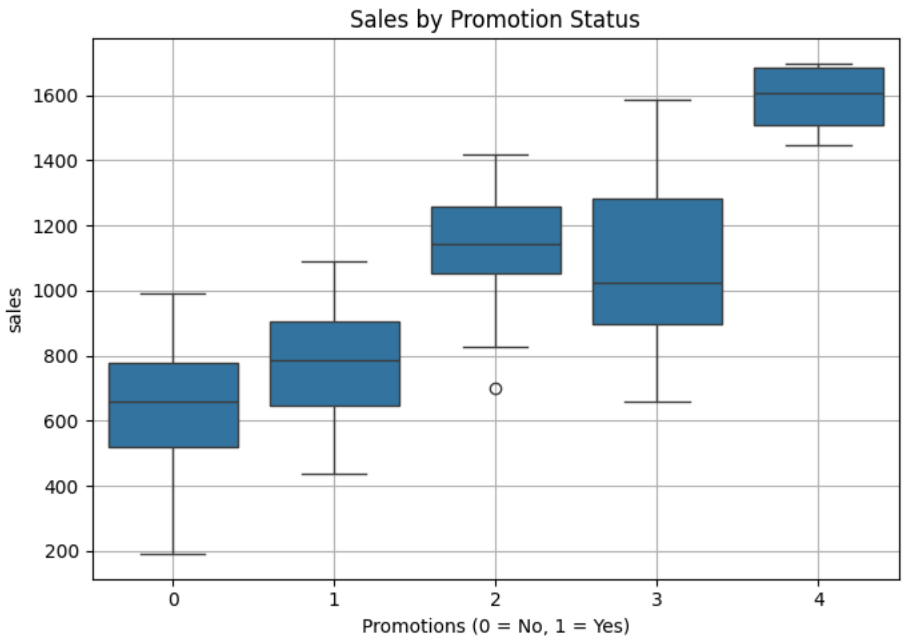
```
# Plotting

# Sales over time
plt.figure(figsize=(14, 6))
plt.plot(df['date'], df['sales'], color='blue')
plt.title('Sales Over Time')
plt.xlabel('Date')
plt.ylabel('Sales')
plt.grid(True)
plt.tight_layout()
plt.show()
```



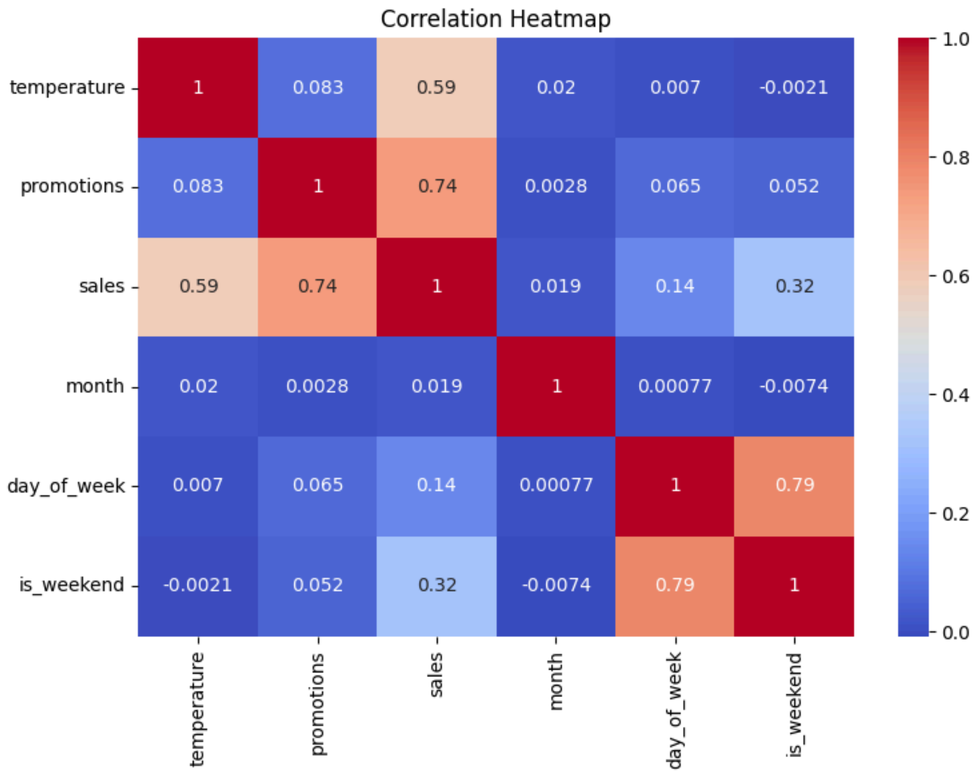

```
# Temperature vs Sales
plt.figure(figsize=(7, 5))
sns.scatterplot(x='temperature', y='sales', data=df)
plt.title('Temperature vs Sales')
plt.grid(True)
plt.tight_layout()
plt.show()
```



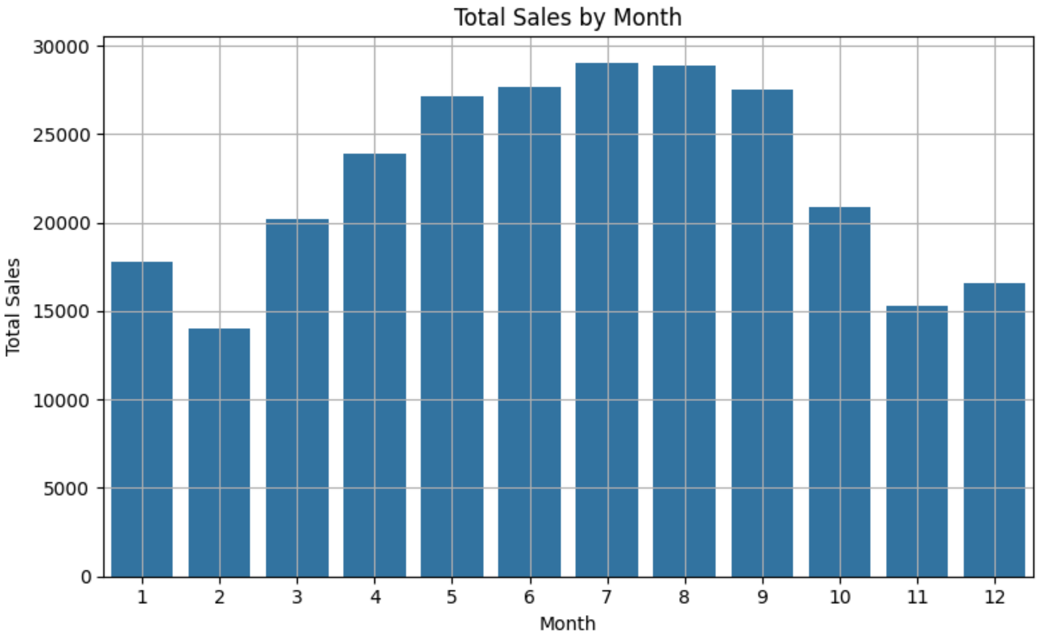

```
# Promotions vs Sales (Boxplot)
plt.figure(figsize=(7, 5))
sns.boxplot(x='promotions', y='sales', data=df)
plt.title('Sales by Promotion Status')
plt.xlabel('Promotions (0 = No, 1 = Yes)')
plt.grid(True)
plt.tight_layout()
plt.show()
```



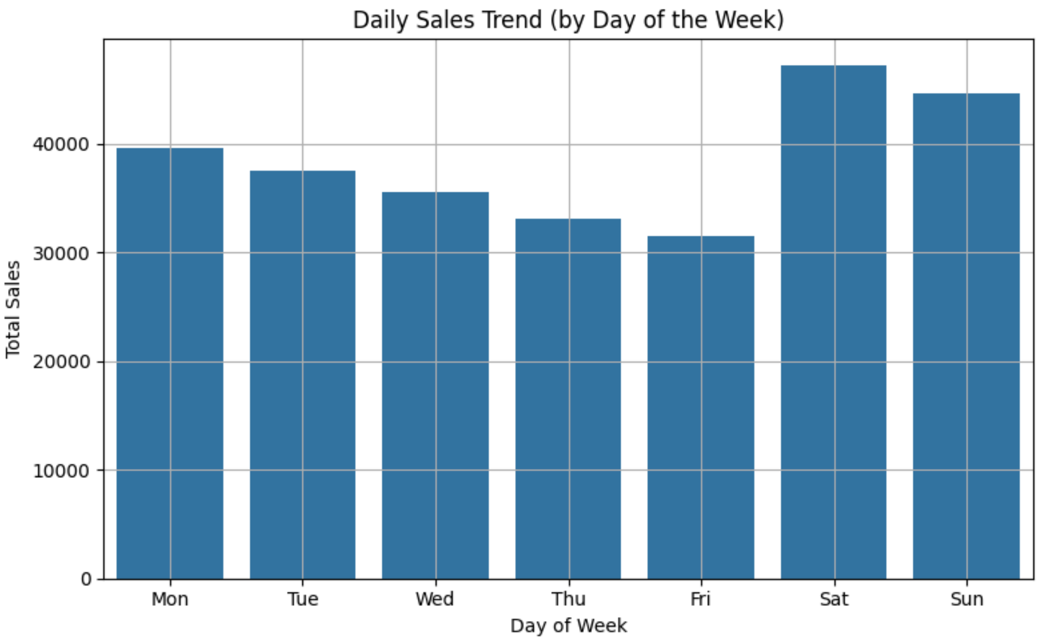

```
# Correlation heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.tight_layout()
plt.show()
```




```
# Monthly Sales Trend
monthly_sales = df.groupby('month')['sales'].sum().reset_index()

plt.figure(figsize=(8, 5))
sns.barplot(x='month', y='sales', data=monthly_sales)
plt.title('Total Sales by Month')
plt.xlabel('Month')
plt.ylabel('Total Sales')
plt.grid(True)
plt.tight_layout()
plt.show()
```

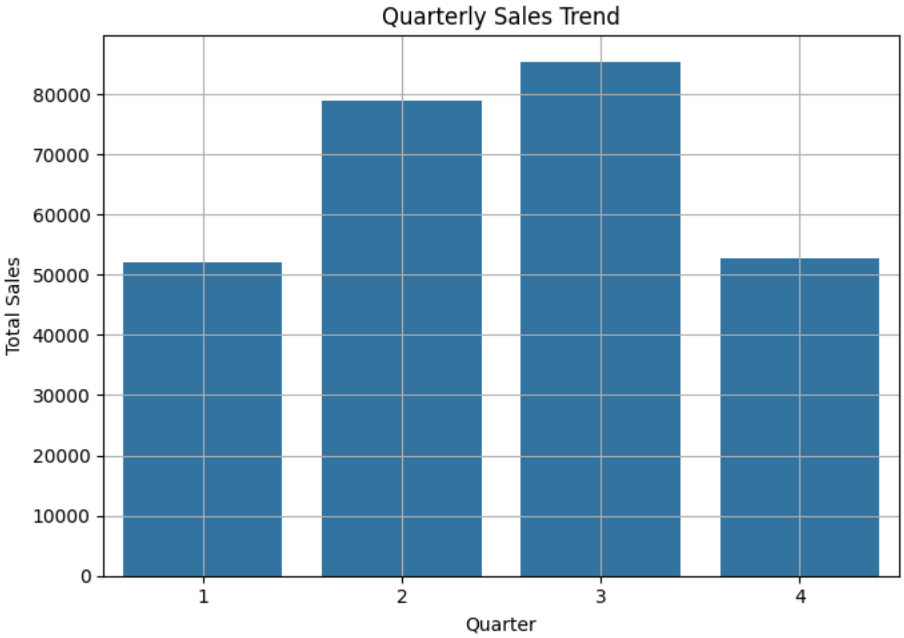



```
# Daily Sales Trend (Day of the Week)
# Mapping day numbers to names
day_map = {0: 'Mon', 1: 'Tue', 2: 'Wed', 3: 'Thu', 4: 'Fri', 5: 'Sat', 6: 'Sun'}
df['day_name'] = df['day_of_week'].map(day_map)

daily_sales = df.groupby('day_name')['sales'].sum().reindex(['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']).reset_index()

plt.figure(figsize=(8, 5))
sns.barplot(x='day_name', y='sales', data=daily_sales)
plt.title('Daily Sales Trend (by Day of the Week)')
plt.xlabel('Day of Week')
plt.ylabel('Total Sales')
plt.grid(True)
plt.tight_layout()
plt.show()
```




```
# Quarterly Sales Trend
df['quarter'] = df['date'].dt.quarter
quarterly_sales = df.groupby('quarter')['sales'].sum().reset_index()

plt.figure(figsize=(7, 5))
sns.barplot(x='quarter', y='sales', data=quarterly_sales)
plt.title('Quarterly Sales Trend')
plt.xlabel('Quarter')
plt.ylabel('Total Sales')
plt.grid(True)
```

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




```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

Time Series decomposition and seasonality analysis

Nischal Pradhan

```
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.seasonal import seasonal_decompose
```

```
df = pd.read_csv("./Preprocessed.csv")
```

```
df['date'] = pd.to_datetime(df['date'])
df = df.sort_values('date')
df.set_index('date', inplace=True)
```

```
df.head()
```

date	temperature	promotions	sales	month	day	year
2022-01-01	-4.01	0	528	1	1	2022
2022-01-02	-5.27	0	424	1	2	2022
2022-01-03	-3.70	1	671	1	3	2022
2022-01-04	-1.93	1	640	1	4	2022
2022-01-05	-5.43	1	520	1	5	2022

```
df.tail()
```

	temperature	promotions	sales	month	day	year
date						
2022-12-27	-3.91	0	377	12	27	2022
2022-12-28	-1.90	0	419	12	28	2022
2022-12-29	-5.20	0	265	12	29	2022
2022-12-30	-4.19	0	237	12	30	2022
2022-12-31	-3.62	0	520	12	31	2022

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 365 entries, 2022-01-01 to 2022-12-31
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   temperature  365 non-null    float64
 1   promotions   365 non-null    int64  
 2   sales        365 non-null    int64  
 3   month        365 non-null    int64  
 4   day          365 non-null    int64  
 5   year         365 non-null    int64  
dtypes: float64(1), int64(5)
memory usage: 20.0 KB
```

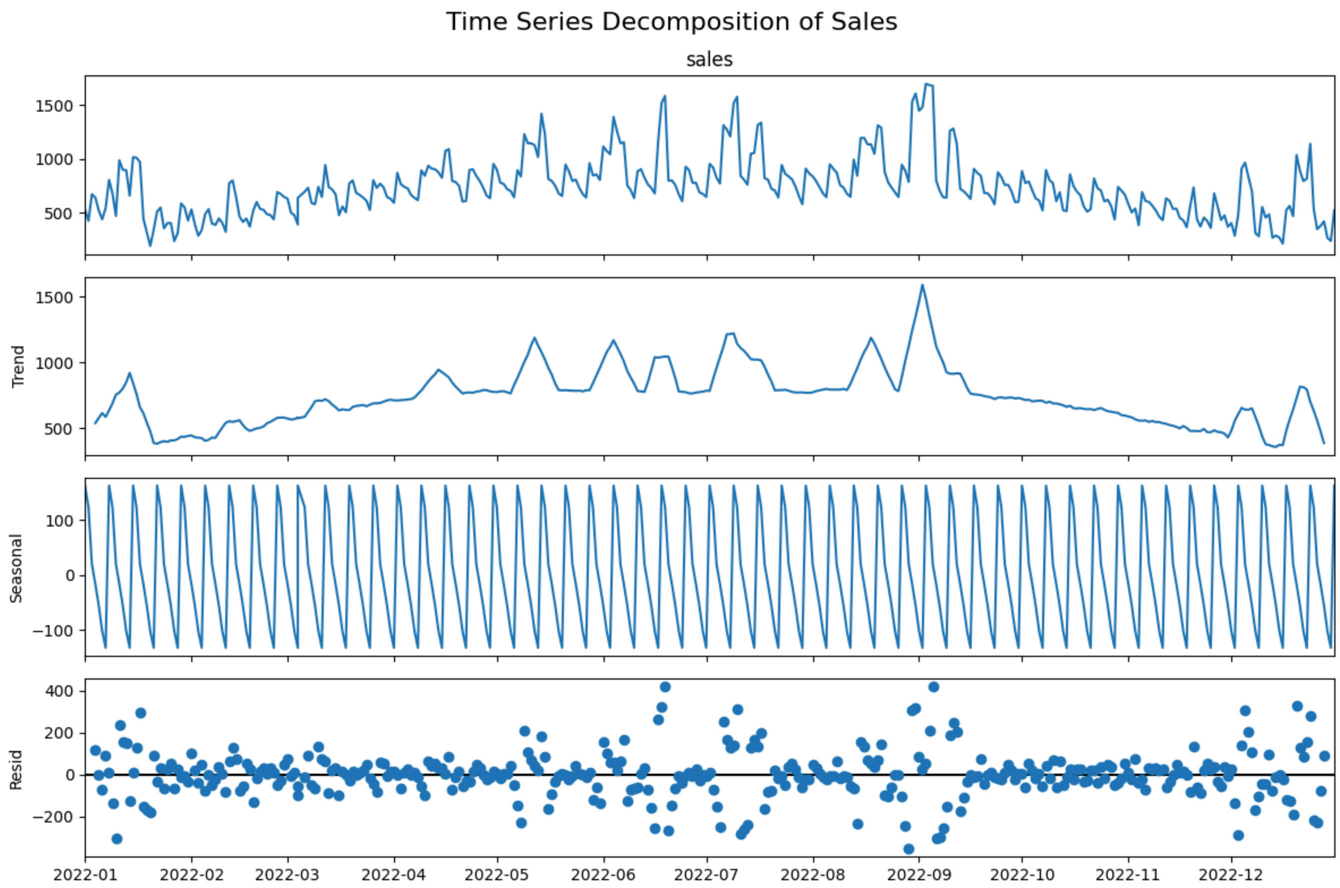
```
df.describe()
```

	temperature	promotions	sales	month	day	year
count	365.000000	365.000000	365.000000	365.000000	365.000000	365.0
mean	10.019890	0.471233	736.972603	6.526027	15.717808	2022.0
std	10.890047	0.993036	275.646810	3.452584	8.811820	0.0
min	-8.450000	0.000000	190.000000	1.000000	1.000000	2022.0
25%	-0.320000	0.000000	552.000000	4.000000	8.000000	2022.0
50%	9.820000	0.000000	705.000000	7.000000	16.000000	2022.0
75%	20.380000	0.000000	864.000000	10.000000	23.000000	2022.0
max	31.170000	4.000000	1698.000000	12.000000	31.000000	2022.0

Time series decomposition

- Trend: shows long term rise or fall in sales trend over time.
- Seasonality: shows weekly (7-days) and monthly (30-days) sales pattern in regular interval.
- Residuals: shows day to day fluctuation which are not explained by trend and seasonality

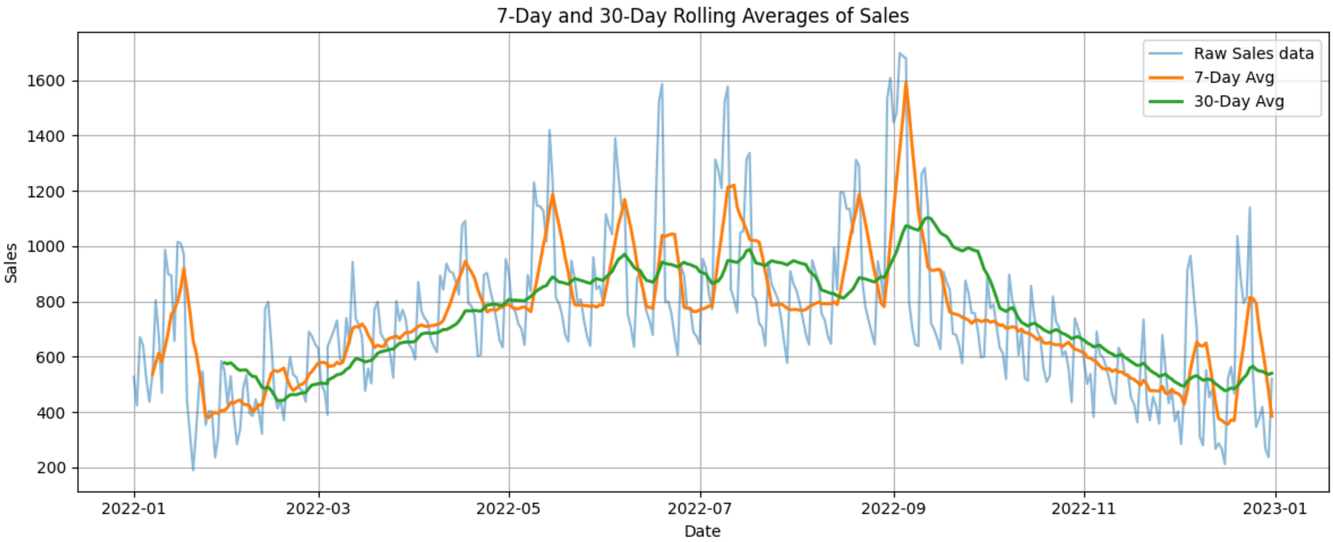
```
# decomposing the sales series into trend, seasonality, residual
result = seasonal_decompose(df['sales'], model='additive', period=7)

fig = result.plot()
fig.set_size_inches(12, 8)
plt.suptitle("Time Series Decomposition of Sales", fontsize=16)
plt.tight_layout()
plt.show()
```



- From trend we can see that the sales trend is increasing massively in mid-year, and is less in the early and end of year
- From Seasonal we can see that weekly sales pattern are almost consistance
- From residuals we can see sudden spikes or outliers through the scatter plot

Sales Trend with 7-Day and 30-Day Rolling Averages

```
# Compute 7-day and 30-days rolling averages
df['7d_avg'] = df['sales'].rolling(window=7).mean()
df['30d_avg'] = df['sales'].rolling(window=30).mean()

plt.figure(figsize=(12, 5))
plt.plot(df['sales'], label='Raw Sales data', alpha=0.5)
plt.plot(df['7d_avg'], label='7-Day Avg', linewidth=2)
plt.plot(df['30d_avg'], label='30-Day Avg', linewidth=2)
plt.title("7-Day and 30-Day Rolling Averages of Sales")
plt.xlabel("Date")
plt.ylabel("Sales")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



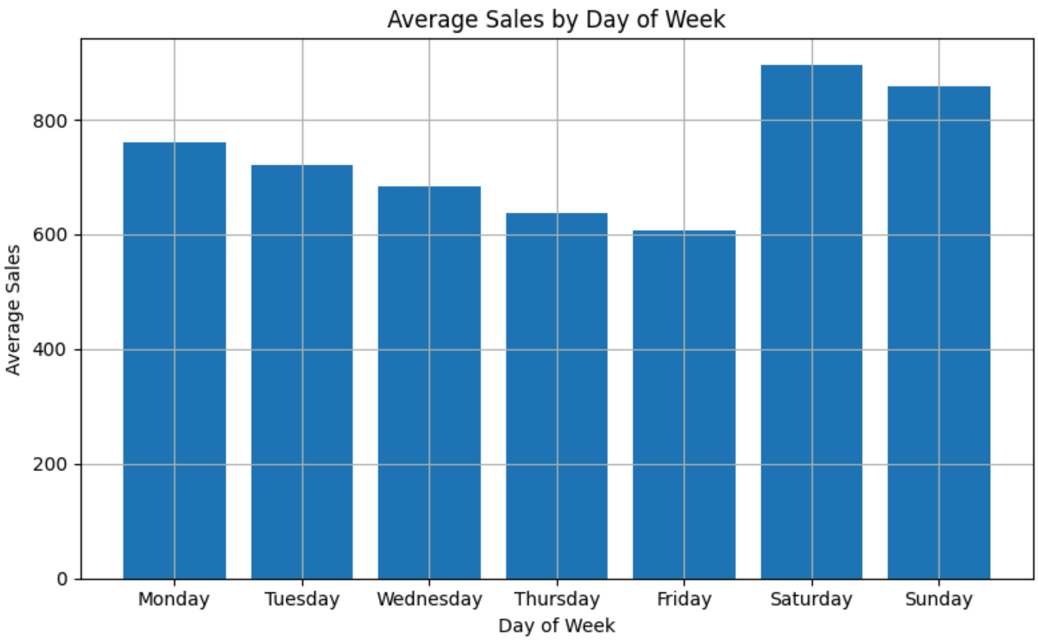
- The line chart shows long term "30-days" and short term "7-days" trends in sales pattern
- The daily sales is highly fluctuating over time and after smoothing clear seasonal patterns and trends can be observed

```
df.describe()
```

	temperature	promotions	sales	month	day	year	7d_avg	30d_avg
count	365.000000	365.000000	365.000000	365.000000	365.000000	365.0	359.000000	336.000000
mean	10.019890	0.471233	736.972603	6.526027	15.717808	2022.0	741.692002	750.835516
std	10.890047	0.993036	275.646810	3.452584	8.811820	0.0	214.898153	178.960819
min	-8.450000	0.000000	190.000000	1.000000	1.000000	2022.0	356.428571	441.133333
25%	-0.320000	0.000000	552.000000	4.000000	8.000000	2022.0	579.642857	577.675000
50%	9.820000	0.000000	705.000000	7.000000	16.000000	2022.0	732.285714	771.266667
75%	20.380000	0.000000	864.000000	10.000000	23.000000	2022.0	843.785714	904.866667
max	31.170000	4.000000	1698.000000	12.000000	31.000000	2022.0	1590.428571	1103.400000

Average sales per day

```
df['day_name'] = df.index.day_name()
day_avg = df.groupby('day_name')['sales'].mean().reindex([
    'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'
])

plt.figure(figsize=(8, 5))
plt.bar(day_avg.index, day_avg.values)
plt.title('Average Sales by Day of Week')
plt.xlabel('Day of Week')
plt.ylabel('Average Sales')
plt.grid(True)
plt.tight_layout()
plt.show()
```



- Sales are high on the weekends
- On the weekdays, sales are gradually decreasing, being lowest on Friday

Linear Regression

Author: Arish Panjwani

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
```

```
#import data
df = pd.read_csv("Preprocessed-v2.csv")
```

```
#Ensure columns are numeric
df["temperature"] = pd.to_numeric(df["temperature"], errors='coerce')
df["promotions"] = pd.to_numeric(df["promotions"], errors='coerce')
df["sales"] = pd.to_numeric(df["sales"], errors='coerce')
```

```
# Drop missing values
df.dropna(subset=["temperature", "promotions", "sales"], inplace=True)
```

```
df.shape
```

```
(365, 9)
```

Define feature and target variable

```
X = df[["temperature", "promotions"]]
```

```
y = df["sales"]
```

Train Test Split

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
```

Fit Model

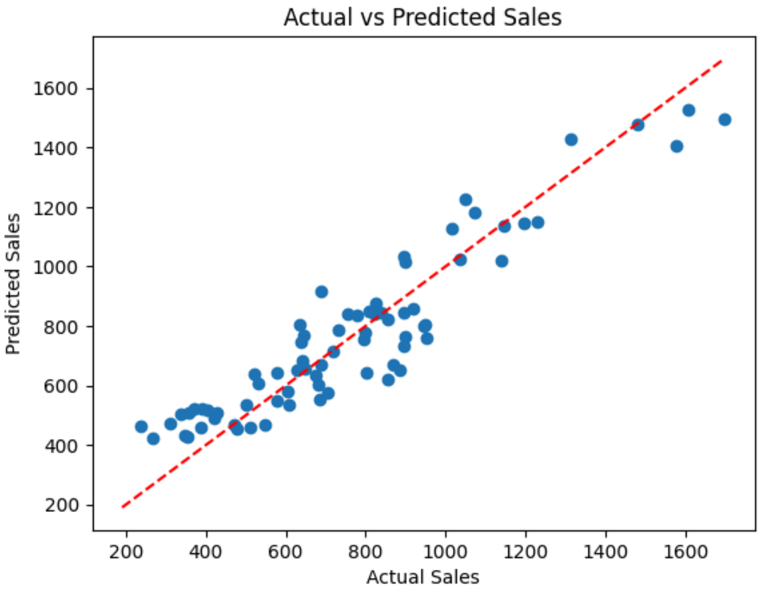
```
model = LinearRegression()
model.fit(X_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

Predict Data

```
y_pred = model.predict(X_test)
```

```
plt.scatter(y_test, y_pred)
plt.xlabel("Actual Sales")
plt.ylabel("Predicted Sales")
plt.title("Actual vs Predicted Sales")
plt.plot([y.min(), y.max()], [y.min(), y.max()], "r--")
plt.show()
```




```
# Assume y_test and y_pred are already defined from your model
r2 = r2_score(y_test, y_pred)

# Treat R² as a percentage accuracy
accuracy = r2 * 100

print(f"Model Accuracy (based on R² score): {accuracy:.2f}%")
```

```
Model Accuracy (based on R² score): 86.74%
```

Findings from My Model:

- I used a dataset with 365 rows with columns like temperature, promotions, and sales.

- The aim was to predict sales based on temperature and whether there was any promotion running.

- First, I cleaned the data a bit by making sure all values are numbers and removed missing stuff.

- Then I splitted the data into training and testing – 80% for train and 20% for test.

- Model Accuracy(based on R^2 score): 86.74%

Model Evaluation

Author: Jeffin John Abraham

```
# Evaluation of Linear Regression
intercept = model.intercept_
coefficients = model.coef_
r2 = r2_score(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test, y_pred)
```

```
print(f"Intercept: {intercept:.2f}")
print(f"Coefficients: {coefficients}")
print(f"R² Score: {r2:.2f}")
print(f"MSE (Mean Squared Error): {mse:.2f}")
print(f"RMSE (Root Mean Squared Error): {rmse:.2f}")
print(f"MAE (Mean Absolute Error): {mae:.2f}")
```

```
Intercept: 516.72
Coefficients: [ 13.1219723  191.16411222]
R² Score: 0.87
MSE (Mean Squared Error): 13211.34
RMSE (Root Mean Squared Error): 114.94
MAE (Mean Absolute Error): 95.73
```

Predictive Modeling: Random Forest

Author: Kanika .

Objective

To build a Random Forest model that predicts daily sales based on: temperature, promotions, Month, Day.

This helps understand the influence of different factors on sales and improve decision-making regarding promotions and seasonal planning.

Why Random Forest?

- Handles nonlinear relationships and feature interactions well
- Robust to overfitting with proper tuning
- Provides feature importance for interpretability

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
import matplotlib.pyplot as plt
```

```
# Loading dataset
df = pd.read_csv("Preprocessed-v2.csv")
```

```
# The relevant features
df = df[["temperature", "promotions", "Month", "Day", "sales"]].dropna()
```

```
# Features and target
X = df[["temperature", "promotions", "Month", "Day"]]
y = df["sales"]
```

```
# Train/test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Training the model
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
```

```
RandomForestRegressor
RandomForestRegressor(random_state=42)
```

```
y_pred = model.predict(X_test)
```

Evaluation

```
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
print(f"RMSE (Root Mean Squared Error): {rmse:.2f}")
```

```
RMSE (Root Mean Squared Error): 113.06
```

```
r2 = r2_score(y_test, y_pred)
print(f"R² Score: {r2:.2f}")
```

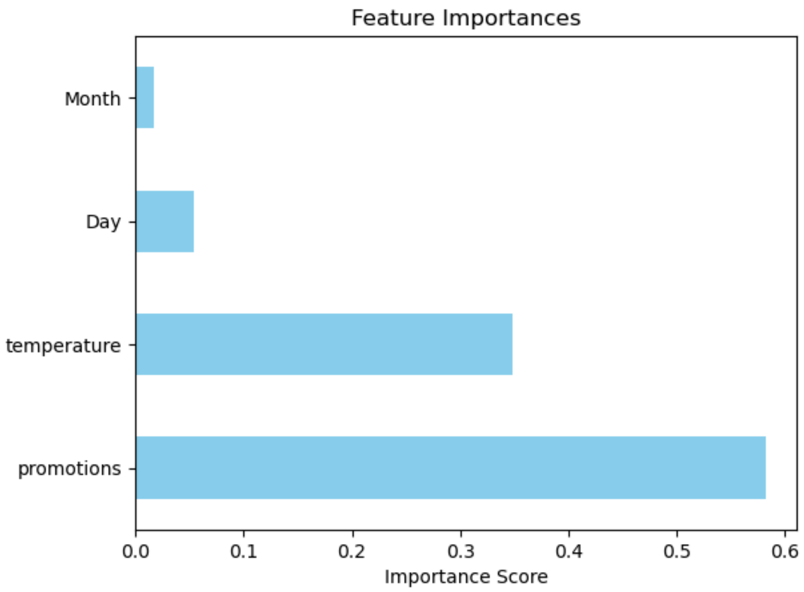
```
R² Score: 0.85
```

```
mae = mean_absolute_error(y_test, y_pred)
print(f"MAE (Mean Absolute Error): {mae:.2f}")
```

```
MAE (Mean Absolute Error): 90.58
```

```
# Feature importance
importance = pd.Series(model.feature_importances_, index=X.columns).sort_values(ascending=False)
```

```
# Plot for feature importance
importance.plot(kind="barh", title="Feature Importances", color="skyblue")
plt.xlabel("Importance Score")
plt.show()
```



Insights

- The most important feature for predicting sales is **promotions**, followed by **temperature**.
- The model explains a significant portion of variance in sales (R^2) and the error metrics indicate reasonable prediction accuracy.
- Random Forest is not as interpretable as linear regression.
- It does not provide direct coefficients or causal insights.

Conclusion Random Forest was a suitable choice for capturing nonlinearities and interactions. Further improvements can be made by tuning hyperparameters or incorporating additional features.

Hyperparameter Tuning (Random Forest)

Author: Ashish Lama

```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
import seaborn as sns
import matplotlib.pyplot as plt
```

```
df = pd.read_csv("Preprocessed-v2.csv")
```

```
df = df[["temperature", "promotions", "Month", "Day", "sales"]].dropna()
```

```
X = df[["temperature", "promotions", "Month", "Day"]]
y = df["sales"]
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Defining parameters
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [5, 10, None],
    'min_samples_split': [2, 5],
    'max_features': ['sqrt', 'log2']
}
```

```
# Applying GridSearchCV
grid_search = GridSearchCV(
    estimator=RandomForestRegressor(random_state=42),
    param_grid=param_grid,
    cv=5,
    scoring='r2',
    n_jobs=-1
)
grid_search.fit(X_train, y_train)
```

```
GridSearchCV
best_estimator_:
RandomForestRegressor
RandomForestRegressor
```

```
# Tuned model evaluation
tuned_model = grid_search.best_estimator_
y_pred_tuned = tuned_model.predict(X_test)
```

```
print("Best Params:", grid_search.best_params_)
```

```
Best Params: {'max_depth': 5, 'max_features': 'sqrt', 'min_samples_split': 2, 'n_estimators': 100}
```

Evaluation

```
mse = mean_squared_error(y_test, y_pred_tuned)
rmse = np.sqrt(mse)
print(f"RMSE (Root Mean Squared Error): {rmse:.2f}")
```

```
RMSE (Root Mean Squared Error): 117.97
```

```
r2 = r2_score(y_test, y_pred_tuned)
print(f"R² Score: {r2:.2f}")
```

```
R² Score: 0.84
```

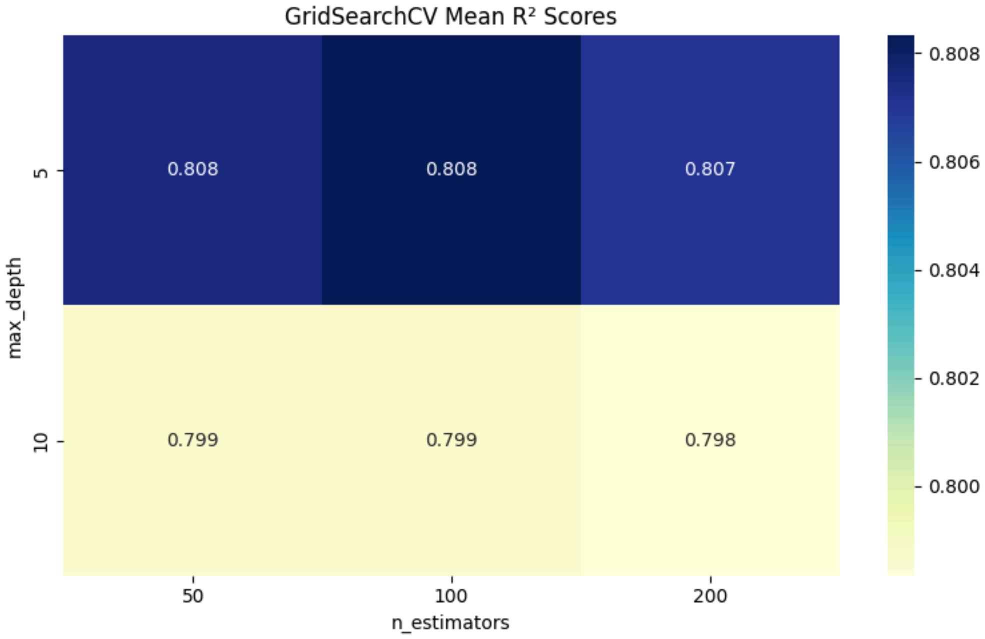
```
mae = mean_absolute_error(y_test, y_pred_tuned)
print(f"MAE (Mean Absolute Error): {mae:.2f}")
```

```
MAE (Mean Absolute Error): 95.11
```

```
results_df = pd.DataFrame(grid_search.cv_results_)
pivot_table = results_df.pivot_table(
    values="mean_test_score",
    index="param_max_depth",
    columns="param_n_estimators",
    aggfunc="mean"
)
print(pivot_table)
```

```
param_n_estimators       50        100       200
param_max_depth
5                   0.807504  0.808322  0.807099
10                  0.798699  0.798742  0.798357
```

```
plt.figure(figsize=(8, 5))
sns.heatmap(pivot_table, annot=True, fmt=".3f", cmap="YlGnBu")
plt.title("GridSearchCV Mean R² Scores")
plt.xlabel("n_estimators")
plt.ylabel("max_depth")
plt.tight_layout()
plt.show()
```



Insights

From the heatmap, we can see that max_depth = 5 consistently gave better R^2 scores across all n_estimators. This suggests that deeper trees may be overfitting or not contributing significantly.

n_estimators = 100 with max_depth = 5 gave the best performance ($R^2 \approx 0.84$), meaning 100 trees struck a good balance between bias and variance.

Performance dropped slightly or plateaued at 200 estimators, implying diminishing returns with more trees.

sqrt for max_features worked better than log2, indicating that considering fewer features per split helped reduce overfitting.

Random Forest Interpretability

Author: Devanshi Adhikari

```
import pandas as pd
import shap
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
```

```
df = pd.read_csv('Preprocessed-v2.csv')
```

```
print(df.columns)
```

```
Index(['date', 'temperature', 'promotions', 'sales', 'Unnamed: 4', 'Date',
       'Month', 'Day', 'Yeat'],
      dtype='object')
```

```
df = df[['temperature', 'promotions', 'sales']].dropna()
```

```
# Convert promotions column to numeric
df['promotions'] = pd.to_numeric(df['promotions'], errors='coerce')
```

```
df = df.dropna(subset=['promotions', 'temperature', 'sales'])
```

```
X = df[['temperature', 'promotions']]
y = df['sales']
```

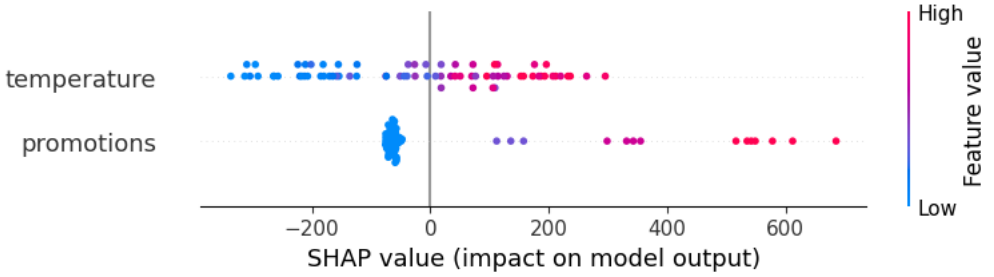
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
model = RandomForestRegressor(random_state=42)
model.fit(X_train, y_train)
```

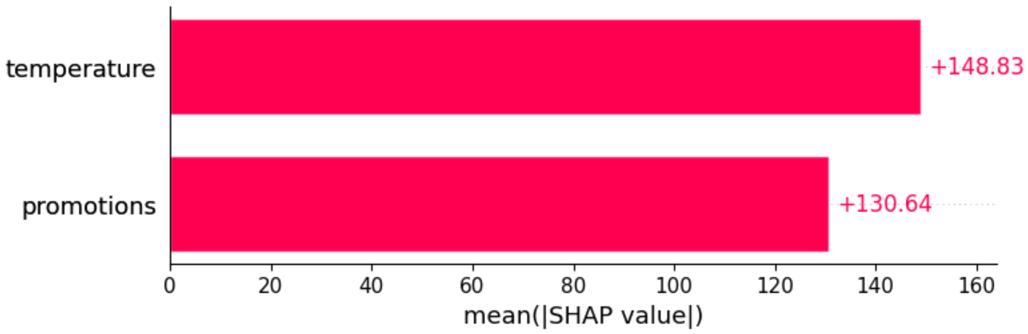
```
RandomForestRegressor
RandomForestRegressor(random_state=42)
```

```
explainer = shap.Explainer(model, X_train)
shap_values = explainer(X_test, check_additivity=False)
```

```
shap.summary_plot(shap_values, X_test)
```




```
shap.plots.bar(shap_values)
```



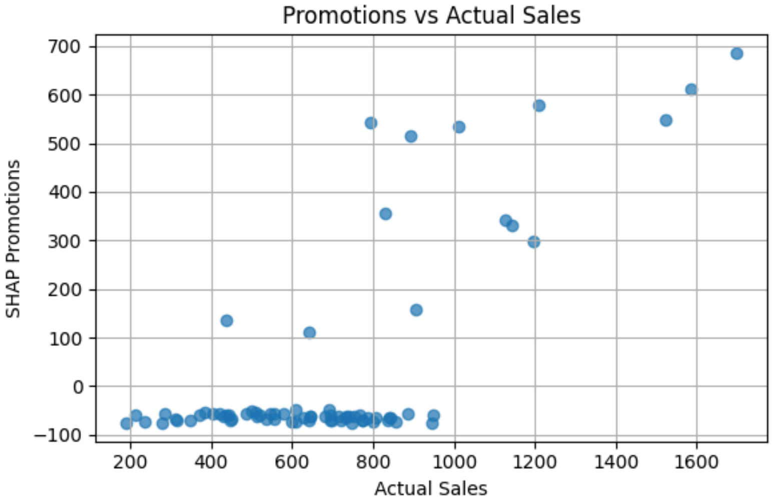

```
print(df.head())
```

```
   temperature  promotions  sales
0        -3.62           0    520
1        -4.19           0    237
2        -5.20           0    265
3        -1.90           0    419
4        -3.91           0    377
```

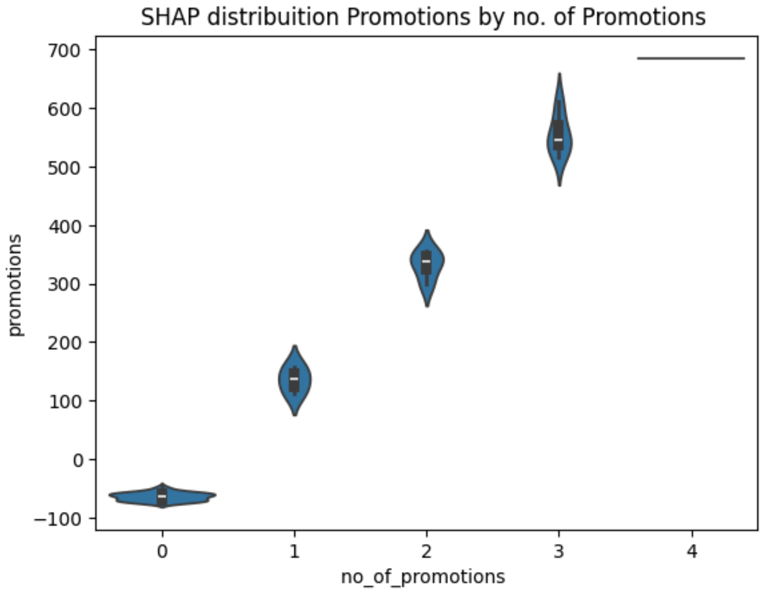
```
df.columns = df.columns.str.strip().str.lower()
```

```
shap_df = pd.DataFrame(shap_values.values, columns=X_test.columns)
sales_actual = y_test.reset_index(drop=True)
shap_promotions = shap_df['promotions']
```

```
plt.figure(figsize=(6, 4))
plt.scatter(sales_actual, shap_promotions, alpha=0.7)
plt.xlabel("Actual Sales")
plt.ylabel("SHAP Promotions")
plt.title("Promotions vs Actual Sales")
plt.grid(True)
plt.tight_layout()
plt.show()
```




```
import seaborn as sns

X_test_copy = X_test.copy()
X_test_copy['no_of_promotions '] = X_test_copy['promotions'].astype(int)
shap_df['no_of_promotions '] = X_test_copy['no_of_promotions '].values

sns.violinplot(data=shap_df, x="no_of_promotions ", y="promotions")
plt.title("SHAP distribuition Promotions by no. of Promotions ")
plt.show()
```



K-Means Clustering

Author: Advait Pandit

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import seaborn as sns
from sklearn.decomposition import PCA
```

```
df=pd.ExcelFile("Dl_Data.xlsx")
```

```
# Sheet-1
df=df.parse('Sheet1')
```

```
#Convert numeric columns
df=df.copy()
df['sales']=pd.to_numeric(df['sales'],errors='coerce')
df['temperature']=pd.to_numeric(df['temperature'],errors='coerce')
df['promotions']=pd.to_numeric(df['promotions'],errors='coerce')
```

```
#Drop the empty NA rows
df.dropna(subset=['sales','temperature','promotions'],inplace=True)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 363 entries, 0 to 364
Data columns (total 4 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   date         363 non-null    datetime64[ns]
 1   temperature  363 non-null    float64
 2   promotions   363 non-null    float64
 3   sales        363 non-null    float64
dtypes: datetime64[ns](1), float64(3)
memory usage: 14.2 KB
```

```
#Daily Clustering
```

```
#Select essential columns and removing all we don't want for standardization Kmeans algorithm
daily_data=df[['sales','temperature','promotions']]
scaler=StandardScaler()
daily_scaled=scaler.fit_transform(daily_data)
```

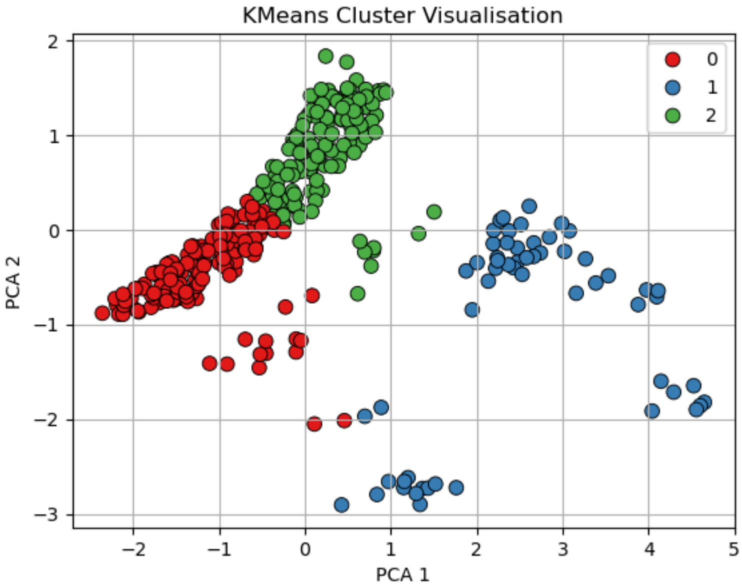
```
#we will use k=3 that is generally consider as best fit.
kmeans=KMeans(n_clusters=3,random_state=42,n_init=10)
df['cluster']=kmeans.fit_predict(daily_scaled)
```

```
C:\Users\ADVAIT\anaconda3\envs\tf-gpu\lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarning: KMeans is known to
  warnings.warn(
```

```
#See days in each cluster
print(df['cluster'].value_counts())
```

```
cluster
0    156
2    150
1     57
Name: count, dtype: int64
```

```
# Apply PCA to scaled daily data
pca=PCA(n_components=2)
pca_result=pca.fit_transform(daily_scaled)
df['pca1']=pca_result[:, 0]
df['pca2']=pca_result[:, 1]
sns.scatterplot(data=df,x='pca1',y='pca2',hue='cluster',palette='Set1',s=60,edgecolor='black')
plt.title("KMeans Cluster Visualisation")
plt.xlabel("PCA 1")
plt.ylabel("PCA 2")
plt.grid(True)
plt.legend()
plt.show()
```




```
# Analyze average values per cluster
df.groupby('cluster')[['sales','temperature','promotions']].mean()
```

cluster	sales	temperature	promotions
0	523.661743	0.252500	0.102564
1	1202.121850	14.373509	2.596491
2	782.493546	18.344733	0.053333

```
#Monthy Clustering
```

```
# Group by month and calculate average sales, temperature, promotions
df['month']=df['date'].dt.month
monthly_data=df.groupby('month').agg({
    'sales':'mean',
    'temperature':'mean',
    'promotions':'mean'
}).reset_index()

print(monthly_data)
```

```
    month        sales  temperature  promotions
0       1   573.167369    -4.734516    0.903226
1       2   500.761859    -0.992500    0.107143
2       3   652.237629     5.750968    0.096774
3       4   797.055465    13.314000    0.233333
4       5   876.913689    19.970645    0.451613
5       6   923.660246    24.785667    0.766667
6       7   953.612379    24.702069    0.793103
7       8   931.198749    20.640000    0.709677
8       9   916.446318    13.914000    0.866667
9      10   673.181312     6.529677    0.000000
10     11   509.158143     0.135000    0.000000
11     12   535.898443    -4.370968    0.741935
```

```
#Standardization for monthly data
monthly_features=monthly_data[['sales','temperature','promotions']]
scaler=StandardScaler()
monthly_scaled=scaler.fit_transform(monthly_features)
```

```
#In Kmeans cluster we will use k=3 consider as generally best fit
kmeans_monthly=KMeans(n_clusters=3,random_state=42,n_init=10)
monthly_data['cluster']=kmeans_monthly.fit_predict(monthly_scaled)
```

```
C:\Users\ADVAIT\anaconda3\envs\tf-gpu\lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarning: KMeans is known to
  warnings.warn(
```

```
#CLuster allcation as per months
monthly_data.sort_values(by='cluster')
```

	month	sales	temperature	promotions	cluster
1	2	500.761859	-0.992500	0.107143	0
2	3	652.237629	5.750968	0.096774	0
3	4	797.055465	13.314000	0.233333	0