Sales Forecasting Across Multiple Retail Stores

BY:-PRIYANSHU KUMAR

DATE:- JUNE 2025

Agenda

- PROJECT OVERVIEW
- DATA ANALYSIS
- EDA

I. Sales before and after holidays
II. Average Month Sale
III. Correlation Between Customer And Sales
IV. Sales During Prome and Sales During Non-Promo
V. Sales Per Day(Monday - Saturday)

- ML Model
- Feature Importance
- Residual Analysis
- Deep Learning
- HTML , app.py

Project Overview

The main objective of the project was to **forecast daily sales for their retail stores across multiple cities up to six weeks in advance**. Traditionally, store managers were making sales predictions based on personal judgment and past experience, which often lacked accuracy.

Data Analysis

Datasets
Historical store sales
Promotions
Holidays
Assortment
Competition.

Key Field

- Sales (Target), Store ID, Date
- Promotions (Promo, Promo2, PromoInterval)
- Holidays (StateHoliday, SchoolHoliday)
- Competition Distance & Opening Info
- StoreType & Assortment

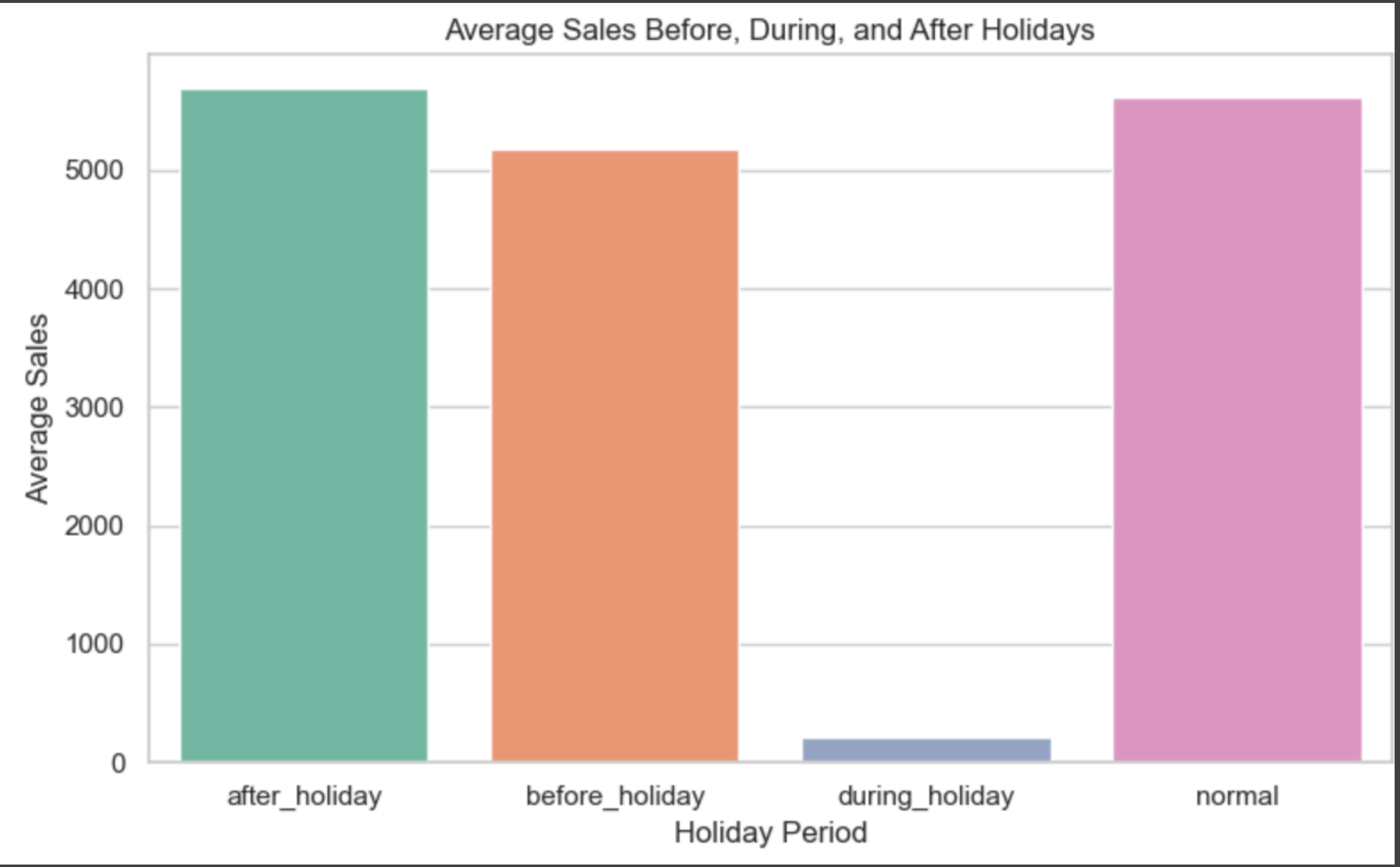
Sales before and after holidays

<u>Lowest Sales During Holidays:</u>
The graph shows that average sales drop significantly during holidays, likely due to store closures or reduced operating hours.

<u>Sales Spike After Holidays:</u>
There's a noticeable increase in sales after holidays, suggesting a post-holiday shopping surge, possibly driven by delayed purchases or post-holiday promotions.

Stable Sales Before Holidays: Sales are relatively high before holidays, potentially due to pre-holiday shopping activity.

<u>Normal Days Perform Well:</u>
Interestingly, normal (non-holiday) days have almost the highest average sales, indicating they form a strong baseline.



Average Month Sale

December records the highest average sales, indicating a strong year-end demand (possibly due to holidays, bonuses, or festive shopping).

July also shows a notable sales peak, possibly driven by mid-year promotions or events.

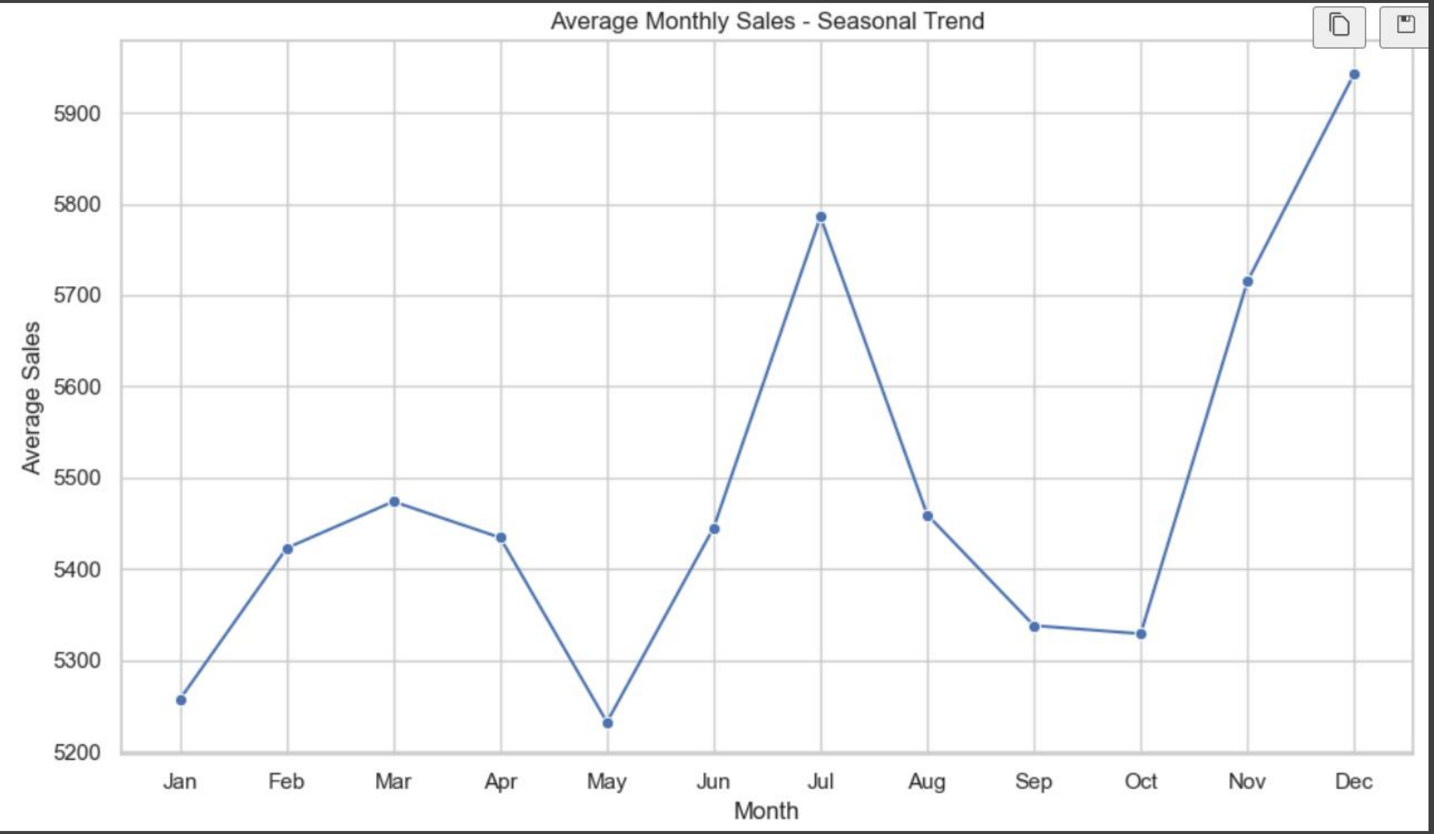
Low Sales Periods:

May has the lowest average sales, which could indicate a seasonal dip—possibly a lull after spring holidays or promotional gaps.

Steady Periods:

Sales are relatively stable from January to April with a slight upward trend.

August to October reflects a minor dip or stagnation in average sales before the sharp rise in November and December.



Correlation Between Customer And Sales

The scatter plot shows a clear upward trend, indicating a strong positive correlation between the number of customers and sales — as the number of customers increases, sales tend to increase as well.

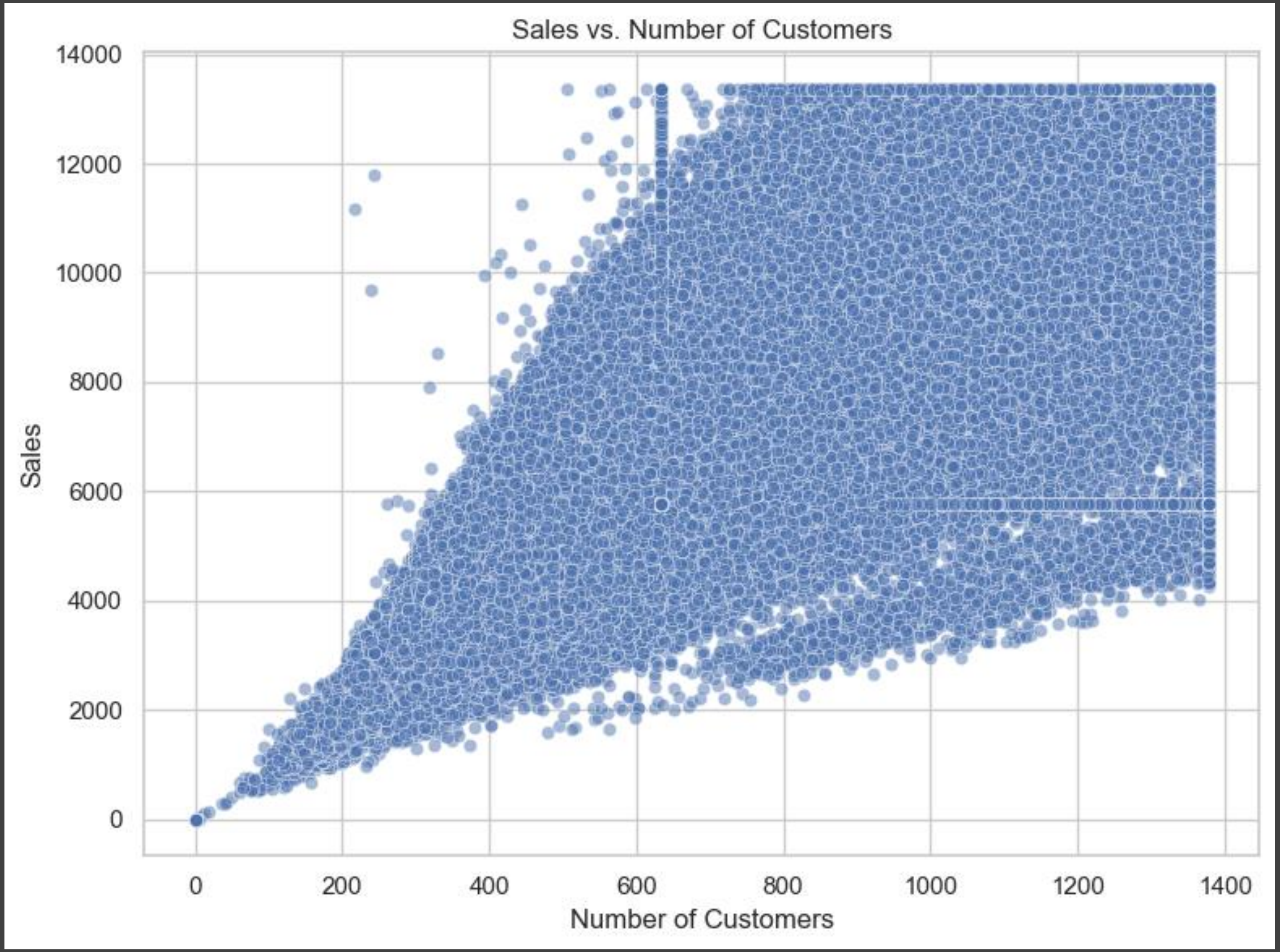
Sales Saturation Zones:

A visible horizontal band appears around 6000 in sales, regardless of customer count, indicating a possible price cap or product limit (e.g., fixed price items or promotion thresholds).

Similarly, many values are concentrated at specific customer counts (like 600, 1200), possibly indicating store-specific capacities or reporting artifacts.

Wide Spread at Higher Customer Counts:

While low customer numbers correspond to low sales, higher customer numbers show more variability in sales, suggesting variation in purchasing behavior per customer (e.g., not all customers buy the same quantity/value).

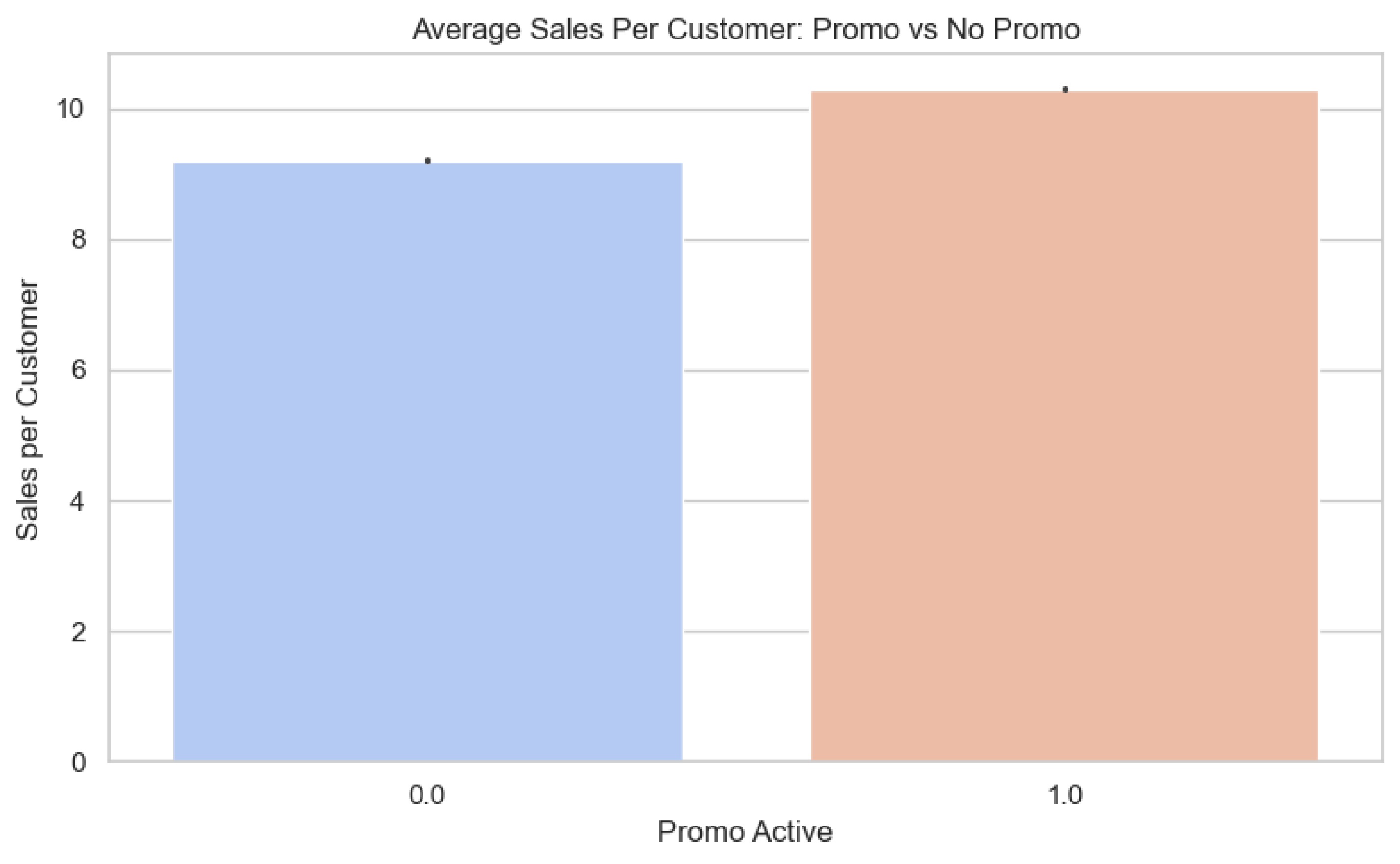


Sales During Prome and Sales During Non-Promo

The presence of a promotion positively influences customer spending behavior, indicating that promotions are effective at encouraging customers to spend more per visit.

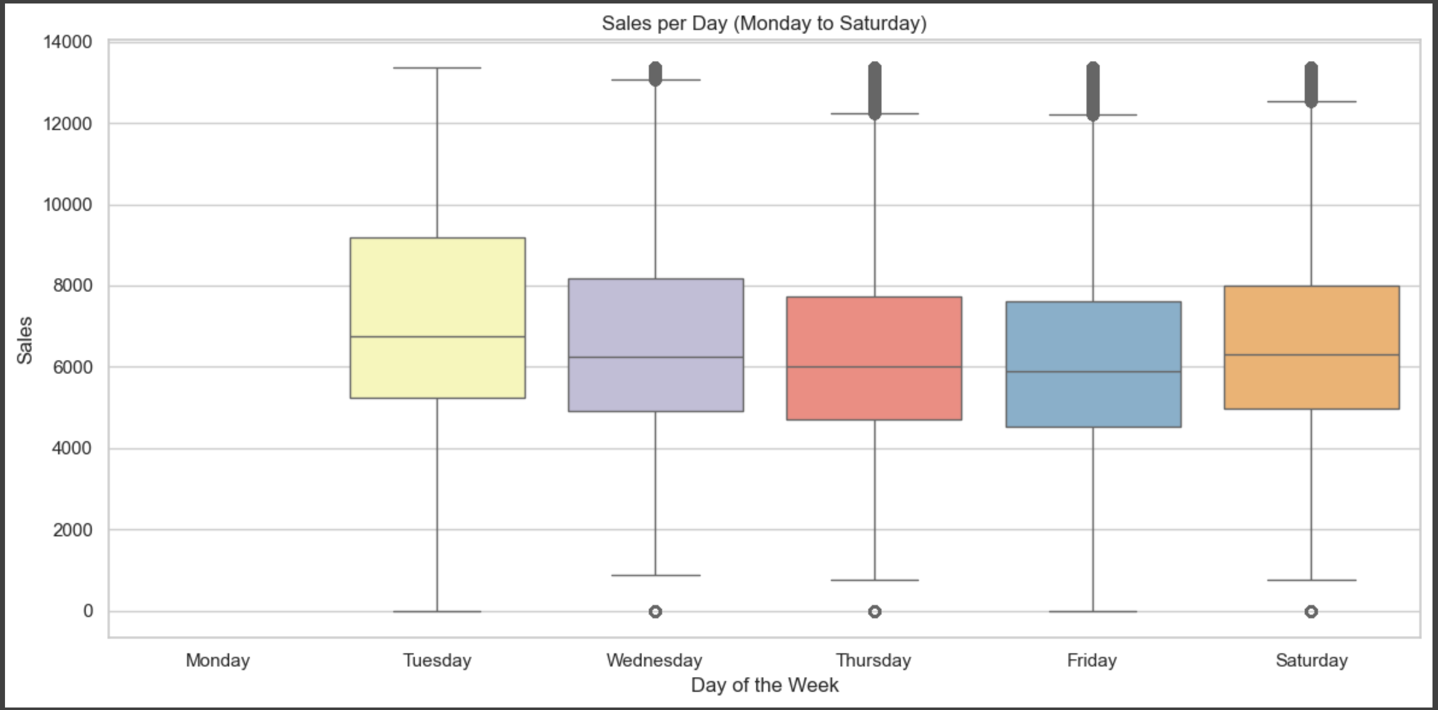
Conclusion:

Promotions lead to a noticeable uplift in per-customer sales, making them a valuable tool for driving revenue. This supports the inclusion of the Promo variable as an important feature in any predictive modeling or marketing decision-making.



Sales Per Day(Monday - Saturday)

- **Sales dip sharply during holidays**, likely due to store closures or reduced footfall.
- **Post-holiday sales spike**, suggesting customer return and promotional push.
- **Normal days consistently outperform holiday-related days**, reinforcing the impact of holidays on revenue.
- **Recommendation**: Plan promotions immediately after holidays and explore strategies to keep select stores open during low-activity holidays.



ML Modeling

RMSE (Root Mean Squared Error): 1233.95

- Measures the average difference between predicted and actual values
- Squares errors before averaging (so larger errors are penalized more)
- In this case, the model's predictions are typically off by about $1,234
- Lower is better (0 would be perfect)

MAE (Mean Absolute Error): 818.42

- Similar to RMSE but without squaring the errors
- Represents the average absolute difference between predictions and reality
- Here, predictions are off by about $818 on average
- Often easier to interpret than RMSE

R^2 (R-squared): 0.74

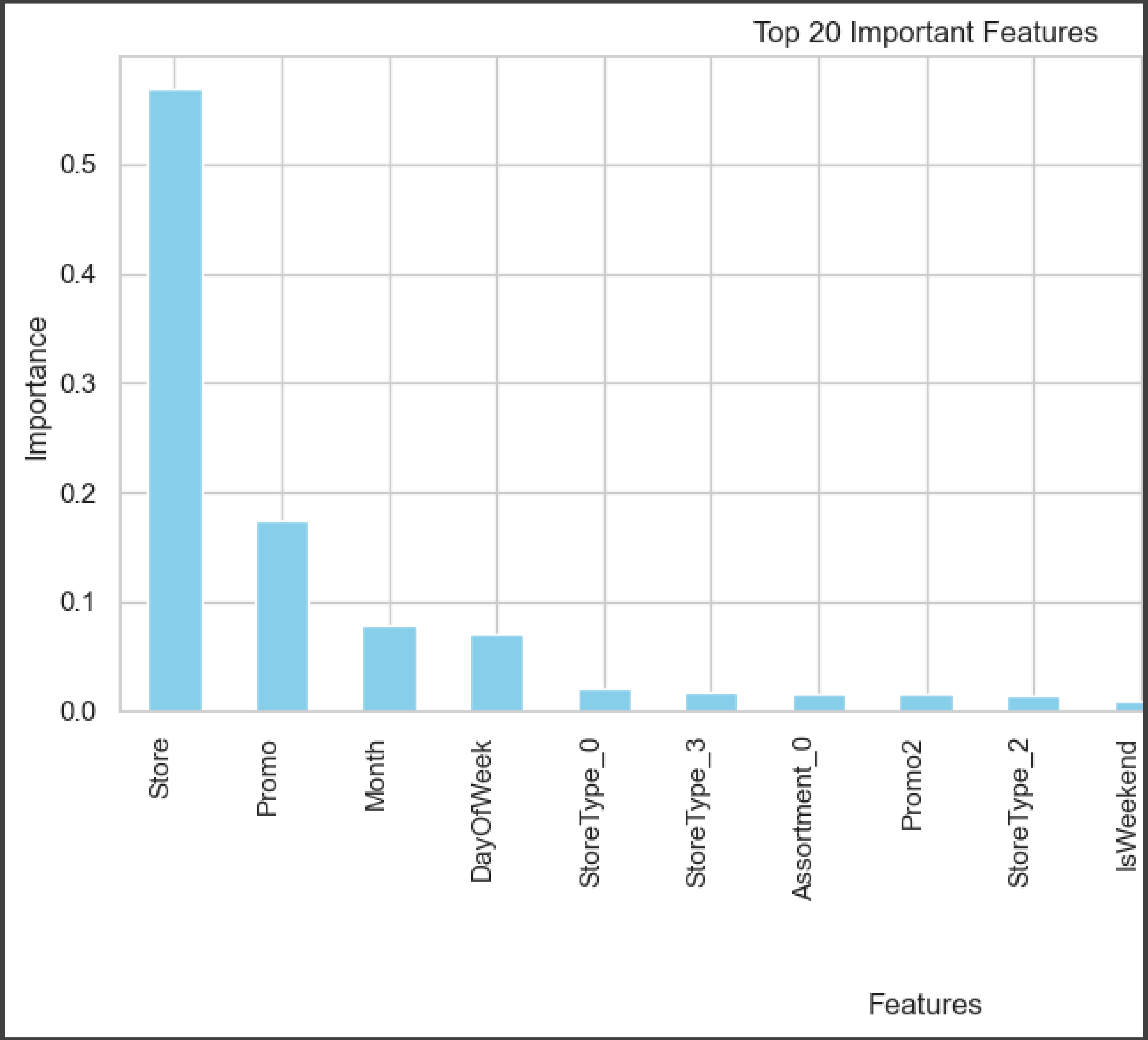
- Shows what percentage of the variation in the data is explained by the model
- Ranges from 0 (no explanation) to 1 (perfect explanation)
- 0.74 means the model explains 74% of the variability - quite good!

```python
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np

y_pred = pipeline.predict(X_val)

# RMSE
rmse = np.sqrt(mean_squared_error(y_val, y_pred))

# MAE
mae = mean_absolute_error(y_val, y_pred)

# R²
r2 = r2_score(y_val, y_pred)

print(f"Validation RMSE: {rmse:.2f}")
print(f"Validation MAE: {mae:.2f}")
print(f"Validation R²: {r2:.2f}")
```

```
Validation RMSE: 1233.95
Validation MAE: 818.42
Validation R²: 0.74
```

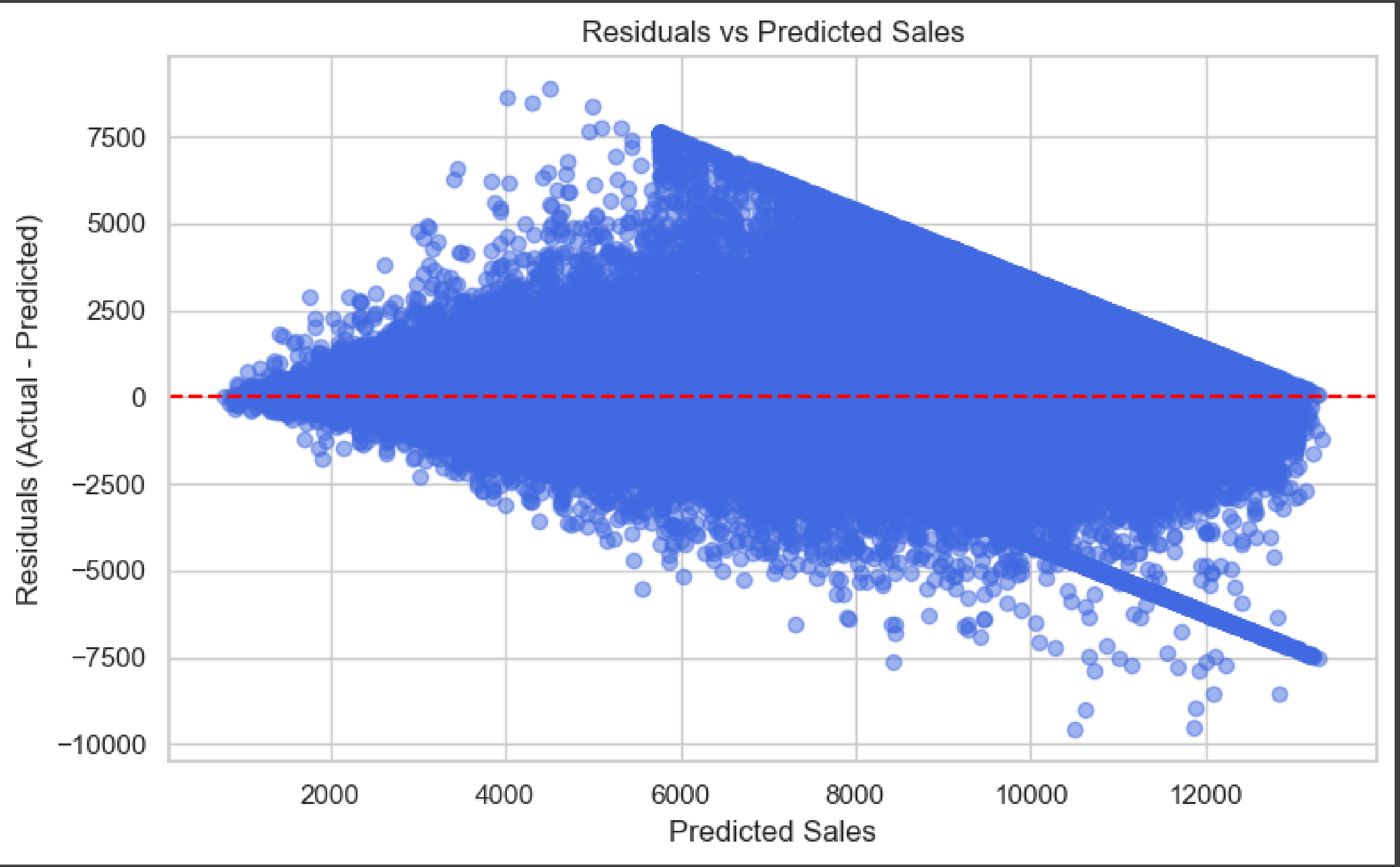
Features Importance

- **Store ID is the most impactful feature**, confirming that each store has distinct sales behavior.
- **Promotions and Month** are strong sales drivers, reflecting seasonality and marketing influence.
- **Day of the Week** impacts buying behavior, with clear weekday/weekend patterns.
- Store characteristics like **StoreType** and **Assortment** contribute meaningfully but less than dynamic features.
- Insights from feature importance can guide targeted promotions and store-level strategies.



Residual Analysis

- Residuals are mostly centered around zero Model has low overall bias
- Indicates need for additional features or different model complexity for high-sales stores.
- Overprediction trend at high sales suggests opportunity for model fine-tuning.
- Residual spread increases with predicted sales Model performs better on low-sales predictions,



Deep Learning

```python
# Feature list used during training
features = ['Store', 'Promo', 'Promo2', 'SchoolHoliday', 'StateHoliday', 'StoreType', 'Assortment',
            'CompetitionDistance', 'Month', 'Year', 'DayOfWeek', 'WeekOfYear', 'IsWeekend']

# Predict
X_test = df_test[features]
test_predictions = pipeline.predict(X_test)
df_test['PredictedSales'] = test_predictions

# Save output
submission = df_test[['Id', 'PredictedSales']].rename(columns={'PredictedSales': 'Sales'})
submission.to_csv('submission.csv', index=False)
print("✅ Submission file created successfully.")
```

```
✅ Submission file created successfully.
```

```python
import joblib
from datetime import datetime
import os
```

```python
##Create Output Directory
# Optional: Save models in a subdirectory
model_dir = "saved_models"
os.makedirs(model_dir, exist_ok=True)
```

```python
##Save Model with Timestamp
# Format: model_YYYY-MM-DD-HH-MM-SS.pkl
timestamp = datetime.now().strftime("%Y-%m-%d-%H-%M-%S")
model_filename = f"rf_pipeline_{timestamp}.pkl"
model_path = os.path.join(model_dir, model_filename)

# Save pipeline
joblib.dump(pipeline, model_path)
print(f"Model saved as: {model_path}")
```

```
Model saved as: saved_models\rf_pipeline_2025-06-14-15-12-48.pkl
```

Deep Learning

```
# Start MLflow run
mlflow.set_experiment("Rossmann_LSTM")
with mlflow.start_run(run_name="lstm_sales_forecasting"):

    model = Sequential([
        LSTM(64, input_shape=(window_size, 1)),
        Dense(1)
    ])
    model.compile(optimizer='adam', loss='mse')

    history = model.fit(X_train, y_train, epochs=20, batch_size=16, validation_data=(X_test, y_test))

    y_pred = model.predict(X_test)
    y_pred_rescaled = scaler.inverse_transform(y_pred)
    y_test_rescaled = scaler.inverse_transform(y_test.reshape(-1, 1))
    rmse = np.sqrt(mean_squared_error(y_test_rescaled, y_pred_rescaled))

    # Log metrics and model
    mlflow.log_param("window_size", window_size)
    mlflow.log_param("epochs", 20)
    mlflow.log_metric("rmse", rmse)
    mlflow.keras.log_model(model, artifact_path="lstm_model")

    print(f"Logged LSTM model with RMSE: {rmse:.2f}")
```

```
Epoch 1/20
38/38 ━━━━━━━━━━━━━━━━━━━━ 4s 31ms/step - loss: 0.1068 - val_loss: 0.0542
Epoch 2/20
38/38 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - loss: 0.0710 - val_loss: 0.0491
Epoch 3/20
38/38 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - loss: 0.0696 - val_loss: 0.0481
Epoch 4/20
38/38 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - loss: 0.0569 - val_loss: 0.0426
Epoch 5/20
38/38 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - loss: 0.0604 - val_loss: 0.0326
Epoch 6/20
38/38 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - loss: 0.0423 - val_loss: 0.0288
Epoch 7/20
38/38 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - loss: 0.0419 - val_loss: 0.0276
Epoch 8/20
38/38 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - loss: 0.0460 - val_loss: 0.0279
```

```
##Train-Test Split
split = int(0.8 * len(X))
X_train, X_test = X[:split], X[split:]
y_train, y_test = y[:split], y[split:]
```

```
##Build the LSTM Model
model = Sequential([
    LSTM(64, activation='tanh', input_shape=(window_size, 1)),
    Dense(1)
])

model.compile(optimizer='adam', loss='mse')
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 64)	16,896
dense (Dense)	(None, 1)	65

Total params: 16,961 (66.25 KB)

Trainable params: 16,961 (66.25 KB)

Non-trainable params: 0 (0.00 B)

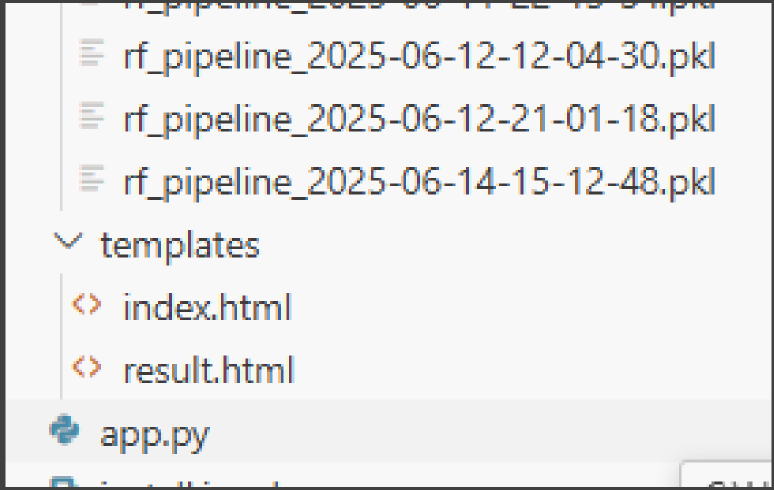
HTML , app.py

app.py > ...

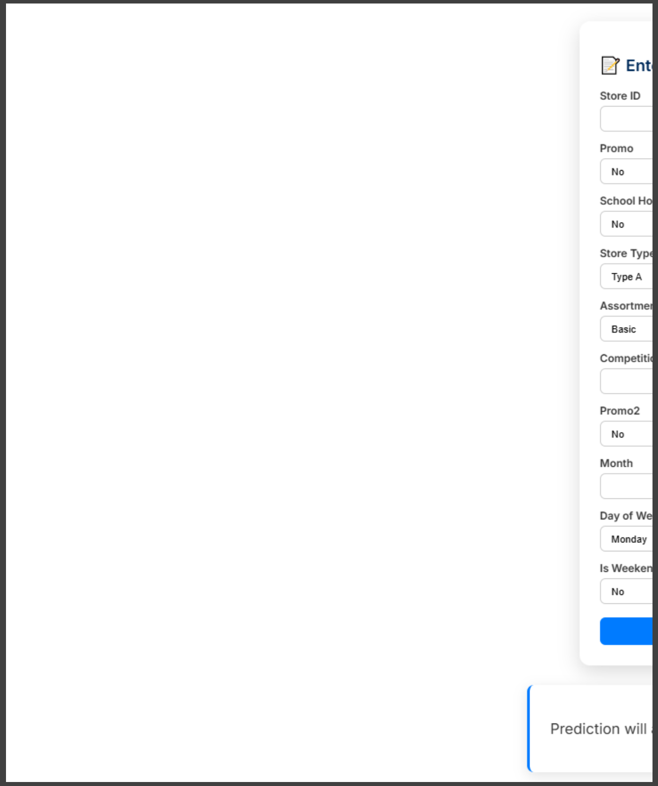
```python
from flask import Flask, request, render_template
import pandas as pd
import joblib  # used for loading the model

app = Flask(__name__)

# Use the full absolute path to your saved model
model_path = r"C:\Users\windows 10\.ipynb_checkpoints\Project6 (1)\
model = joblib.load(model_path)

@app.route('/')
def home():
    return render_template('index.html')

@app.route('/predict', methods=['POST']) # post work as a push in f
def predict():
    try:
        feature_order = [
            'Store', 'Promo', 'SchoolHoliday', 'StoreType', 'Assort
            'CompetitionDistance', 'Promo2', 'Month', 'DayOfWeek',
        ]

        input_values = [float(request.form.get(feature)) for featur
        input_df = pd.DataFrame([input_values], columns=feature_ord
        prediction = model.predict(input_df)[0]

        return render_template('result.html', prediction=round(pred

    except Exception as e:
        return render_template('result.html', prediction=f"Error: {

if __name__ == '__main__':
    app.run(debug=True)
```

templates > index.html > ...

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
  <title>Sales Prediction App</title>
  <link href="https://fonts.googleapis.com/css2?family=Poppins:wght@400;600&display=swap" rel="stylesheet">
  <style>
    body {
      margin: 0;
      font-family: 'Poppins', sans-serif;
      background: linear-gradient(135deg, #e0f7fa, #ffffff);
      display: flex;
      justify-content: center;
      align-items: center;
      min-height: 100vh;
      padding: 20px;
    }

    .container {
      background-color: #ffffff;
      max-width: 1000px;
      width: 100%;
      border-radius: 16px;
      box-shadow: 0 10px 40px rgba(0, 0, 0, 0.1);
      display: flex;
      flex-wrap: wrap;
      overflow: hidden;
    }

    .form-section, .result-section {
```



```
rf_pipeline_2025-06-12-12-04-30.pkl
rf_pipeline_2025-06-12-21-01-18.pkl
rf_pipeline_2025-06-14-15-12-48.pkl
templates
  index.html
  result.html
app.py
```



Thank You