1. Title

Predicting Amazon Sales Using Daily Traffic, Conversion, and Advertising Data with Machine Learning

2. Introduction

Sales outcomes on Amazon can fluctuate significantly based on a variety of factors, such as traffic volume, advertising investment, and conversion rates. In today's highly competitive e-commerce landscape, accurately forecasting sales is crucial for better inventory planning, minimizing stockouts, and optimizing advertising spend. It is especially important to recognize these trends at a daily level to capture subtle shifts that monthly or weekly data could obscure.

This project, undertaken as part of the DAB 422 Capstone Project 2, focuses on using two years of daily Amazon sales, traffic, and advertising data to build a predictive model for total units ordered. By developing a robust machine learning solution, we aim to create a tool that not only predicts sales trends but also aids sellers in making more informed decisions regarding marketing strategies and inventory control. This work is carried out by a team consisting of **Khizer Baig**, **Dhyeykumar Sunilbhai Patel**, **Sahil Thapa**, and **Gaurav** under the guidance of Professor **A. Sodiq Shofoluwe**.

---

3. Related Work

Sales forecasting in e-commerce has long been a critical research area. Numerous studies have highlighted the importance of detailed feature selection and temporal granularity in improving model accuracy. For Amazon specifically, traffic metrics (such as Sessions and Page Views) and advertising metrics (such as Ad Spend and Ad Sales) are widely recognized as key predictors of sales performance.

Traditional models such as Linear Regression offer a basic forecasting framework but often fall short in capturing the complex interactions between advertising efforts and customer behavior. More advanced methods like Random Forests and Gradient Boosting Machines (XGBoost, LightGBM) provide better accuracy but pose challenges related to overfitting and interpretability. Furthermore, research emphasizes the difficulty in predicting sales without incorporating external factors like competitor actions, economic trends, and platform changes—variables that were not available in our project.

Our work builds upon these prior findings by applying daily-level granularity, rigorous feature selection through methods such as Correlation Matrix, Chi-Square Test, and Embedding Methods, and using automated machine learning frameworks (PyCaret) to streamline model selection and evaluation.

4. Methods

4.1 Data Source

The primary dataset for this project was collected directly from Amazon Seller Central, covering a 24-month period on a daily granularity. The dataset included key metrics related to sales, traffic, and advertising performance. Specifically, the fields extracted were:

- Total Units Ordered
- Total Sales (Ordered Product Sales)
- Sessions
- Page Views
- Unit Session Percentage (Conversion Rate)
- Ad Spend
- Ad Sales
- Advertising Efficiency Metrics (ACOS, ROAS, TACOS)
- Featured Offer (Buy Box) Percentage

This comprehensive dataset allowed us to closely track customer behavior and advertising impacts on a day-to-day basis, offering a much richer understanding compared to monthly aggregates.

4.2 Data Cleaning and Preparation

Once the raw data was gathered, it underwent multiple cleaning and preparation steps in a Python environment using libraries such as pandas and matplotlib:

- Removed duplicates and standardized column formats.
- Stripped percentage signs (%) and currency symbols ($) from relevant columns and converted them into appropriate numeric types.
- Dropped redundant columns such as 'Order Items' to focus on features relevant to modeling.
- Inspected and handled missing values using interpolation where feasible; fortunately, our dataset was largely complete.

This process ensured consistency in the data and reduced the risk of errors during model training.

4.3 Feature Engineering

While the primary modeling was done using the original set of selected features, we performed extensive feature engineering to explore the data and support exploratory analysis:

- **Revenue per Session** = Total Sales / Sessions
- **Ad Efficiency Ratio** = Ad Sales / Ad Spend
- **Organic Sales Ratio** = (Total Sales - Ad Sales) / Total Sales
- **Buy Box Impact Score** = (Featured Offer Percentage / 100) * Total Sales

These derived features helped during exploratory analysis (such as Chi-Square tests and correlation matrix generation) to understand deeper relationships between traffic, advertising, and sales performance. However, for the final model building, we restricted our inputs to carefully selected core features to avoid redundancy and multicollinearity.

4.4 Exploratory Data Analysis (EDA)

We conducted extensive EDA to better understand the relationships within our dataset:

- **Summary Statistics**: Examined distributions, means, and standard deviations.
- **Correlation Matrix**: Generated a heatmap to visualize inter-feature correlations. It revealed very strong correlations between Total Sales, Units Ordered, and Featured Offer Percentage.
- **Feature Importance Plot**: Using PyCaret, we plotted feature importance across different models.

Based on the EDA findings, we made critical modeling decisions. Notably, we removed **"Featured Offer (Buy Box) Percentage"** from final model training due to its extremely high correlation with Units Ordered and Total Sales, which could have introduced redundancy and overfitting.

Additionally, Chi-Square tests were used to validate feature relevance statistically, though engineered features like Revenue per Session were ultimately excluded from the final feature set used for predictions.

---

5. Modeling Approach

To build and evaluate predictive models, we used PyCaret's Regression Module, which allowed us to quickly compare the performance of multiple algorithms on our dataset. The setup function was initialized with 'Ordered Product Sales' as the target variable.

We compared a variety of regression models, including:

- Huber Regressor
- Linear Regression
- Random Forest Regressor
- Light Gradient Boosting Machine (LightGBM)
- Extreme Gradient Boosting (XGBoost)
- Orthogonal Matching Pursuit (OMP)

During the model comparison, **Huber Regressor** consistently outperformed others across key metrics such as RMSE, MAE, and R^2 score.

Initial testing with the full dataset, which included the 'Featured Offer (Buy Box) Percentage' feature, resulted in an R^2 value of approximately 0.79. However, after refining our features and removing 'Featured Offer' due to its high correlation, the performance of the Huber Regressor improved significantly, achieving an **R^2** score of approximately **0.87**.

The final model was trained using four features: **Ad Sales, Ad Spend, Sessions, and Unit Session Percentage**. This streamlined input set improved generalizability, reduced the risk of overfitting, and provided a more interpretable model.

---

6. Deployment

After successfully training the Huber Regressor model, we deployed it using a lightweight Flask application. The workflow included:

- Saving the trained model using Python's `joblib` library.
- Setting up a Flask app where users could input new daily metrics (Ad Sales, Ad Spend, Sessions, Unit Session %) and receive a real-time prediction of total units ordered.
- Testing the app with sample inputs to validate prediction accuracy and reliability.

This deployment step transformed our machine learning model into an accessible tool that can assist Amazon sellers in making day-to-day operational decisions based on real-time marketing and traffic data.

Further improvements could involve building a front-end dashboard and automating model updates with incoming live data.

7. Results

Our experiments and analyses revealed key insights about the relationships among Amazon traffic, advertising, and sales:

- **Feature Importance**: Unit Session Percentage and Sessions were the most impactful variables, followed closely by Ad Sales and Ad Spend.
- **Model Performance**:
    - Initial R^2 Score (with Buy Box feature): ~0.79
    - Final R^2 Score (after feature optimization): ~0.87
    - RMSE and MAE values showed strong model reliability and low prediction error.
- **Prediction Visualizations**: Residual plots, prediction vs actual scatterplots, and distribution of errors confirmed minimal bias and a well-generalized model.

By removing highly correlated features, cleaning the dataset thoroughly, and focusing on meaningful predictors, we substantially improved predictive performance.

---

8. Discussion

This project reinforced the critical importance of careful data preprocessing, feature selection, and model evaluation in building robust predictive models. Through the exploratory data analysis, we identified strong dependencies among variables, highlighting the risk of multicollinearity in Amazon sales data.

The Huber Regressor's robust performance further validated its appropriateness for datasets prone to minor outliers or irregularities—common in e-commerce environments.

A key observation was that traffic quantity (Sessions) and traffic quality (Unit Session Percentage) together played a pivotal role in sales outcomes. Advertising spend contributed meaningfully but was less impactful compared to organic traffic and conversion rates.

While external factors like competitor pricing and seasonal trends were not included in this project, the model still achieved significant predictive accuracy, demonstrating the potential of internal Amazon metrics alone to drive valuable insights.

---

9. Conclusion

Our project successfully demonstrated that machine learning models can predict daily Amazon sales (units ordered) with high accuracy using internal metrics such as Ad Spend, Ad Sales, Sessions, and Unit Session Percentage. By methodically cleaning the dataset, removing redundancies, conducting deep exploratory analysis, and selecting a robust algorithm (Huber Regressor), we achieved a final R^2 score of approximately 0.87.

Deploying the model through a Flask application allowed us to translate our analytical findings into a practical, user-friendly tool that can assist Amazon sellers in real-time decision-making. The model was successfully validated by predicting a sample sales figure and replicating the same result within the deployed app.

Future Work:

- Integrate external variables such as competitor prices and seasonal trends.
- Experiment with ensemble modeling techniques for potentially higher accuracy.
- Build an automated pipeline to refresh the model periodically with new data.
- Develop a full web dashboard to enhance usability and visualization.

---

10. Contributions

- **Khizer Baig**: Led the entire project including data gathering, data cleaning, feature engineering, exploratory data analysis (EDA), modeling using PyCaret, model evaluation, Flask application deployment, and writing the final report.
- **Dhyeykumar Sunilbhai Patel**: Assisted with feature engineering, heatmap correlation analysis, and contributed to presentation preparation.
- **Sahil Thapa**: Helped in exploratory data analysis, initial feature selection, and preparing parts of the presentation.
- **Gaurav**: Contributed to preliminary research and supported the team during presentations and documentation phases.

---

11. References

- Amazon Business Reports: Daily Sales and Traffic Reports on Amazon. https://sellercentral.amazon.com/business-reports/ref=xx_sitemetric_favb_xx#/dashboard
- PyCaret Documentation: Regression Module and Model Comparison Techniques. https://pycaret.gitbook.io/docs/
- Capstone Final Dataset and Modeling Notebook: Data cleaning, feature engineering, and regression model testing. https://colab.research.google.com/drive/10XvT5NWIjotu475joXs2Ix_kYM0vdhGk
- Flask App Deployment and Model Prediction Interface: Real-time prediction testing and app deployment. https://colab.research.google.com/drive/1DzOp_T5gs9yHa27I1zkFNE56MIgV3IDo#scrollTo=bbc790e4
- Model Training and Evaluation Notebook: Final model retraining and validation after feature optimization. https://colab.research.google.com/drive/10bxJRAPSy0-_SujrCqT0u8SrBPqBQUl9

---

12. Appendices

Appendix A: Sample of Cleaned Dataset

- Displaying a few rows with key columns: Sessions, Ad Spend, Ad Sales, Unit Session Percentage, Ordered Product Sales.

```
[ ] data.head()
```

	Date	Ordered Product Sales	Units Ordered	Page Views	Sessions	Featured Offer (Buy Box) Percentage	Unit Session Percentage	Adspend	Ad sales	ACOS	ROAS	TACOS
0	3/24/2023	189.96	4	144	113	100.00	3.54	25.0	188.0	13.30	7.52	13.16
1	3/25/2023	800.78	22	141	111	97.87	19.82	17.0	60.0	28.33	3.53	2.12
2	3/26/2023	299.89	11	156	121	100.00	9.09	19.0	20.0	95.00	1.05	6.34
3	3/27/2023	1007.71	29	170	132	98.82	21.97	18.0	109.0	16.51	6.06	1.79
4	3/28/2023	988.72	28	160	130	99.38	21.54	22.0	396.0	5.56	18.00	2.23

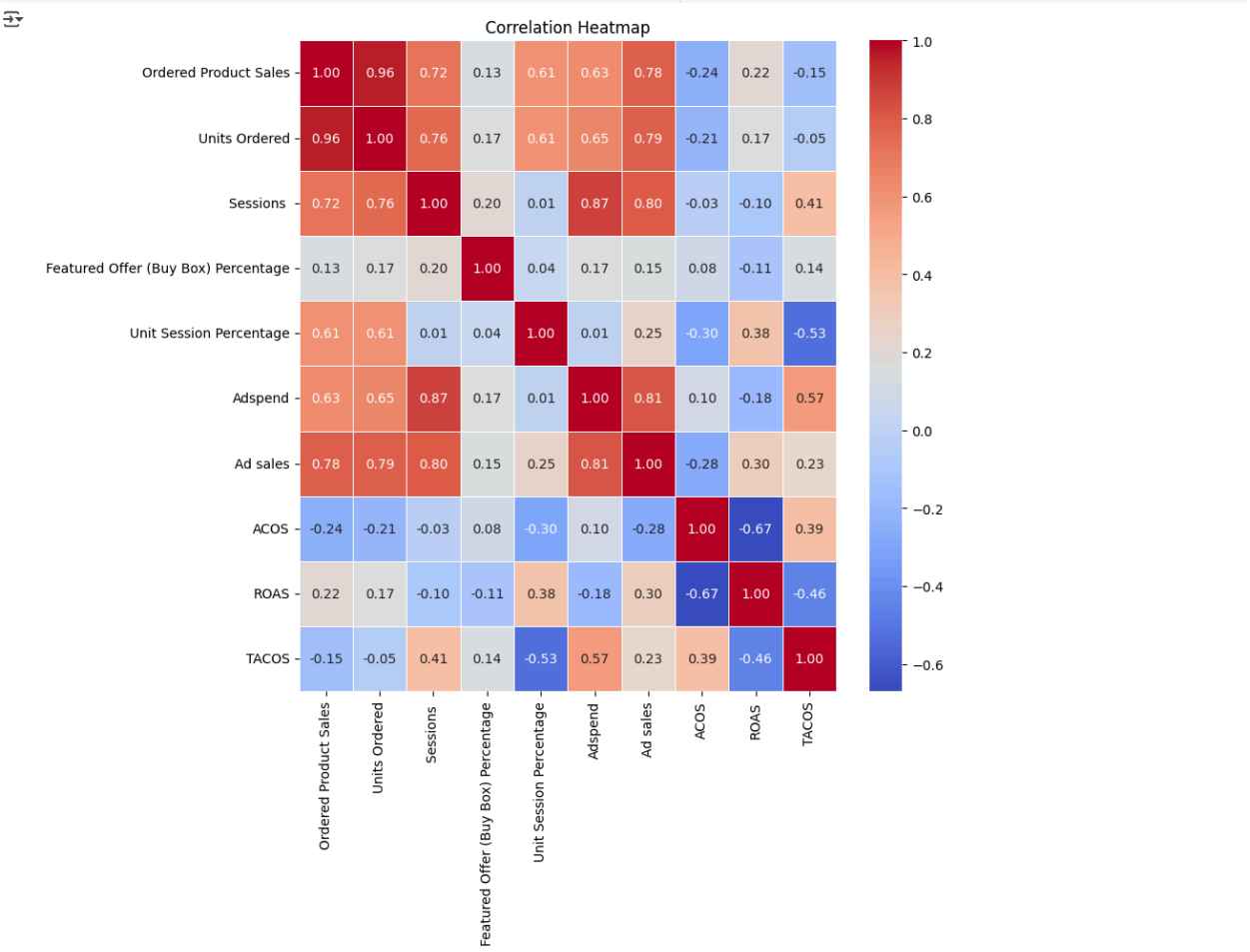
Appendix B: Feature Engineering Formulas

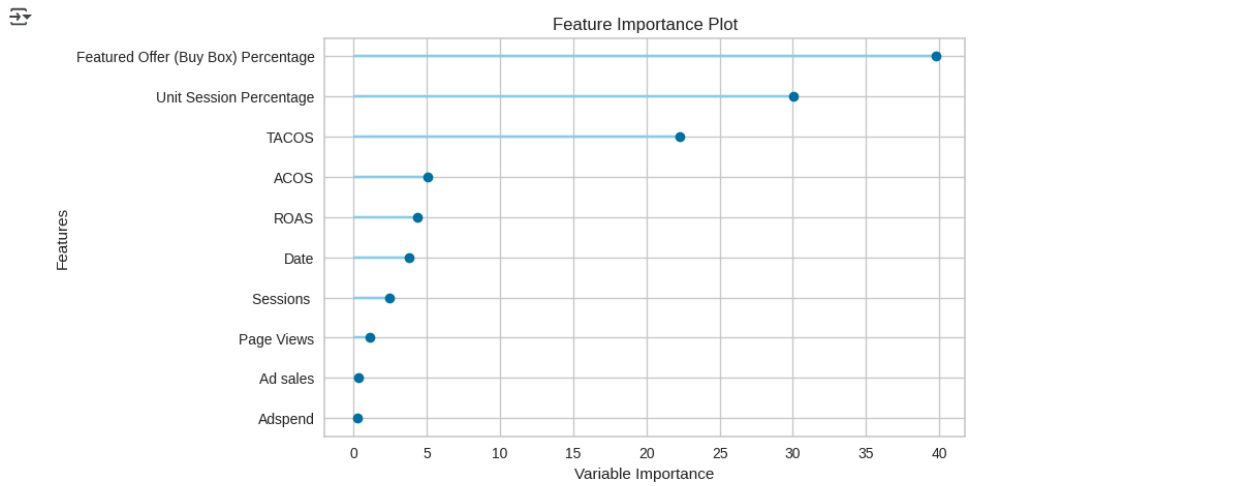
- Revenue per Session = Total Sales / Sessions
- Ad Efficiency = Ad Sales / Ad Spend
- Organic Sales Ratio = (Total Sales - Ad Sales) / Total Sales
- Buy Box Impact = (Featured Offer Percentage / 100) * Total Sales

```
[ ] # Handle missing or non-numeric values before performing arithmetic operations
    df["Sessions - Total"] = pd.to_numeric(df["Sessions - Total"], errors='coerce').fillna(1)  # Avoid division by zero
    df["Total Sales"] = pd.to_numeric(df["Total Sales"], errors='coerce').fillna(0)
    df["Ad Sales"] = pd.to_numeric(df["Ad Sales"], errors='coerce').fillna(0)
    df["Ad Spend"] = pd.to_numeric(df["Ad Spend"], errors='coerce').fillna(1)  # Avoid division by zero
    df["Featured Offer (Buy Box) Percentage"] = pd.to_numeric(df["Featured Offer (Buy Box) Percentage"], errors='coerce').fillna(0)

    # Feature Engineering with missing value handling
    df["Revenue per Session"] = df["Total Sales"] / df["Sessions - Total"]
    df["Ad Efficiency"] = df["Ad Sales"] / df["Ad Spend"]
    df["Organic Sales Ratio"] = (df["Total Sales"] - df["Ad Sales"]) / df["Total Sales"].replace(0, np.nan)  # Avoid division by zero
    df["Buy Box Impact"] = (df["Featured Offer (Buy Box) Percentage"] / 100) * df["Total Sales"]

    # Fill any remaining NaNs after computation
    df.fillna(0, inplace=True)
```

Appendix C: Correlation Matrix and Feature Importance Plots

- Final correlation heatmap illustrating relationships between features.
- Feature importance plot from PyCaret model comparison.





Appendix D: Model Comparison Results (PyCaret Regression Comparison)

- Table showing MAE, MSE, RMSE, R^2 for models like Huber Regressor, LightGBM, Random Forest, etc.

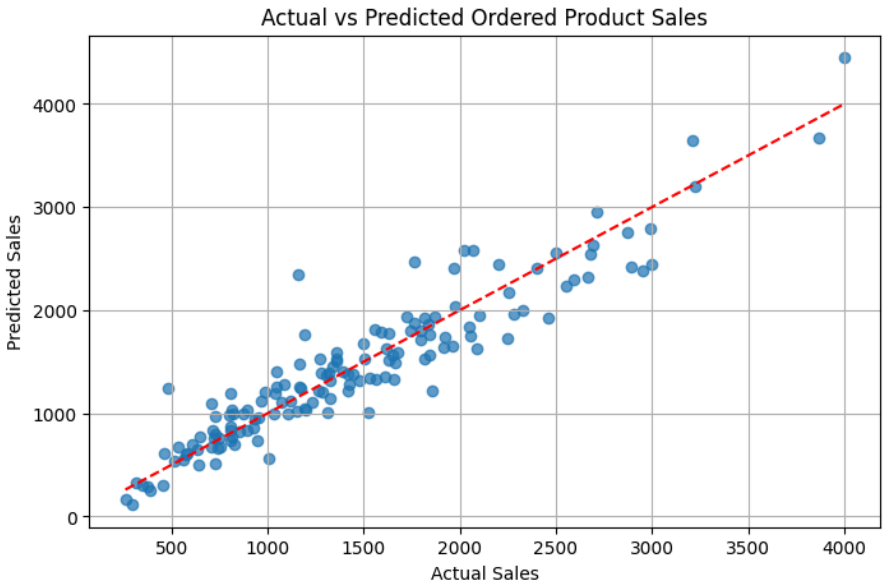
```
[ ] # compare baseline models
    best = compare_models()
```

	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE	TT (Sec)
huber	Huber Regressor	339.5936	203575.7642	445.2365	0.6739	0.3480	0.3233	0.0860
omp	Orthogonal Matching Pursuit	404.4811	285457.3699	528.3497	0.5632	0.4122	0.3780	0.0970
par	Passive Aggressive Regressor	454.7876	372529.3123	603.1342	0.4265	0.4583	0.3638	0.0990
knn	K Neighbors Regressor	504.2476	416685.2094	642.1899	0.3690	0.5213	0.5488	0.0700
lightgbm	Light Gradient Boosting Machine	613.9458	622180.6105	784.9658	0.0694	0.5793	0.6379	1.0300
et	Extra Trees Regressor	617.3819	644442.8975	798.5921	0.0388	0.5837	0.6410	0.3410
xgboost	Extreme Gradient Boosting	632.3877	685868.0969	823.8887	-0.0216	0.5944	0.6499	0.2420
gbr	Gradient Boosting Regressor	633.2943	686049.9850	824.0095	-0.0220	0.5952	0.6528	0.2290
dt	Decision Tree Regressor	634.0188	687889.2680	825.1302	-0.0248	0.5961	0.6542	0.0650
ada	AdaBoost Regressor	631.3812	688488.7638	825.4707	-0.0256	0.5932	0.6441	0.2580
rf	Random Forest Regressor	634.2885	688398.7576	825.4455	-0.0256	0.5961	0.6540	0.4060
lr	Linear Regression	634.4761	688930.7205	825.7396	-0.0263	0.5962	0.6539	1.0400
lasso	Lasso Regression	634.4719	688920.7990	825.7337	-0.0263	0.5962	0.6539	0.0640
ridge	Ridge Regression	634.4745	688927.1636	825.7375	-0.0263	0.5962	0.6539	0.0640
en	Elastic Net	634.4618	688902.8891	825.7229	-0.0263	0.5962	0.6539	0.0640
lar	Least Angle Regression	634.4761	688930.7205	825.7396	-0.0263	0.5962	0.6539	0.0650
llar	Lasso Least Angle Regression	634.4696	688915.5677	825.7306	-0.0263	0.5962	0.6539	0.0880
br	Bayesian Ridge	634.4761	688930.7205	825.7396	-0.0263	0.5962	0.6539	0.1140
dummy	Dummy Regressor	634.4761	688930.7250	825.7397	-0.0263	0.5962	0.6539	0.0640

```
# Clean column names
data.columns = data.columns.str.strip()

# Updated features and target
features = ['Ad sales', 'Adspend', 'Sessions', 'Unit Session Percentage']
target = 'Ordered Product Sales'

X = data[features]
y = data[target]

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

# Initialize and fit Huber Regressor
huber = HuberRegressor()
huber.fit(X_train, y_train)

# Predictions
y_pred = huber.predict(X_test)

# Evaluation
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error:", mse)
print("R² Score:", r2)
```

```
Mean Squared Error: 69357.63550688997
R² Score: 0.8753656199445825
```

```
import matplotlib.pyplot as plt

plt.figure(figsize=(8, 5))
plt.scatter(y_test, y_pred, alpha=0.7)
plt.xlabel("Actual Sales")
plt.ylabel("Predicted Sales")
plt.title("Actual vs Predicted Ordered Product Sales")
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')
plt.grid(True)
plt.show()
```



Appendix E: Flask Application Output Validation

- Predicted Ordered Product Sales = 356.41 units for a sample input.
- Flask app screenshot or log confirming successful prediction matching the trained model.

```
[ ]
    sample_input_df = pd.DataFrame([{
        'Ad sales': 200.0,
        'Adspend': 35.0,
        'Sessions': 120.0,
        'Unit Session Percentage': 12.5
    }])

    predicted_sales = huber.predict(sample_input_df)[0]
    print(f"🔮 Predicted Ordered Product Sales: {predicted_sales:.2f}")
```

```
🔮 Predicted Ordered Product Sales: 356.41
```

```
[ ] import joblib

    # Save the trained model to a file
    joblib.dump(huber, 'huber_sales_model.pkl')
```

```
['huber_sales_model.pkl']
```

```
[ ] from google.colab import files
    files.download('huber_sales_model.pkl')
```