PROJECT TITLE: PREDICTING PROFIT USING TRANSACTION DATA

1. INTRODUCTION AND DATASET DESCRIPTION

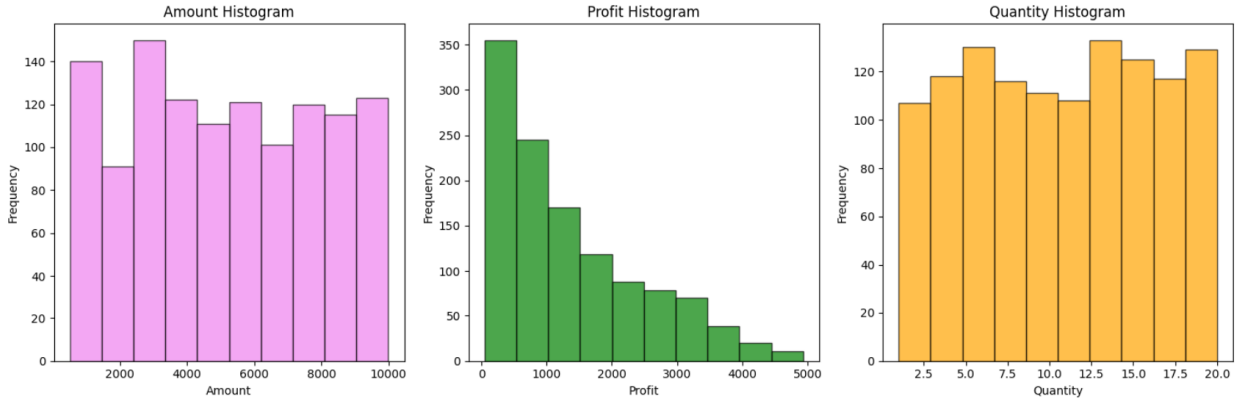
This report details a project focused on predicting the profit generated from individual sales transactions using machine learning and predictive models. Accurately forecasting profit is crucial for businesses to make informed decisions regarding resource allocation, sales strategies, inventory management, and financial planning. By understanding the key drivers of profitability and developing predictive models, businesses can identify high-value opportunities, mitigate risks associated with low-profit transactions, and ultimately improve financial performance.

The analysis utilizes the Sales Dataset.csv dataset, which contains transactional sales records. This dataset includes 1194 rows and 12 columns, capturing details such as Order ID, Amount, Profit, Quantity, Category, Sub-Category, Payment Mode, Order Date, Customer Name, State, and City. The primary goal is to predict Profit based on other relevant features within the dataset. The rationale for this project stems from the business's need to move beyond simple historical reporting towards predictive insights that can proactively guide sales and operational strategies. Understanding which types of transactions yield higher profits allows stakeholders to optimize efforts and forecast more reliably.

2. EXPLORATORY DATA ANALYSIS (EDA)

An initial exploratory data analysis was conducted to understand the dataset's structure, distributions, and relationships. The dataset comprises transactional data with numerical features (Amount, Profit, Quantity) and categorical features (Category, Sub-Category, PaymentMode, State, City). No significant missing values were detected in the core features intended for initial modeling.

Numerical Feature Distributions: Histograms were plotted for the key numerical features: Amount, Profit, and Quantity.

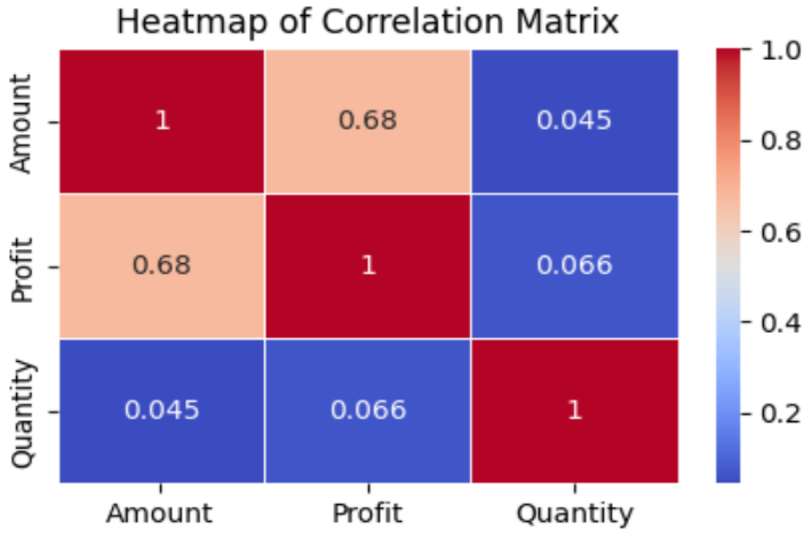


Amount Histogram: The distribution of transaction amounts spans roughly from near 0 up to 10,000. The histogram indicates that transaction amounts are somewhat spread out across this range, without a single strong central peak. There are notable frequencies of transactions in several bins, suggesting a diverse mix of low, medium, and high-value transactions within the dataset.

Profit Histogram: The target variable, Profit, exhibits a wide range and appears to be right-skewed, with most transactions having lower to moderate profits and fewer transactions generating very high profits.

Quantity Histogram: The Quantity of items per transaction ranges from 1 to 20. The distribution appears relatively uniform across this range. Unlike Profit, there isn't a strong skew; transactions with small, medium, and large quantities occur with roughly similar frequencies. This suggests customers purchase a wide variety of items per order.

Numerical Correlations: A heatmap showed a strong positive correlation between Amount and Profit, and moderate positive correlations involving Quantity.



Key Insights from EDA:

Profits are variable and positively correlated with the transaction Amount and Quantity.

The dataset contains useful categorical features (Category, Payment Mode, State) that show different profit characteristics. The data is suitable for regression modeling, aiming to predict the Profit value.

3. DATA CLEANING AND PREPROCESSING

Several preprocessing steps were undertaken to prepare the dataset for machine learning modeling. The rationale behind these steps was to create a clean, numerical dataset suitable for model training and to ensure a robust evaluation process.

Summary of Steps:

- Feature Selection

Columns deemed irrelevant for initial modeling were dropped. This included identifiers (Order ID), high-cardinality text fields (CustomerName, City), potentially redundant date information

(Order Date, Year-Month), and Sub-Category (to simplify the initial model by using the broader Category).

Rationale: This simplifies the models, removes noise, and avoids issues with excessive dimensionality or non-generalizable features. Sub-category was also removed due to high multicollinearity with the Category.

Code snippet:

```
cols_to_drop = ['Order ID', 'CustomerName', 'City', 'Sub-Category', 'Order Date', 'Year-Month']
df.drop(columns=cols_to_drop, inplace=True, errors='ignore')
print(f"Dropped identifier/high-cardinality/unused columns (incl. Sub-Category).")
```

- Handling Data Types

Ensured numerical columns (Amount, Quantity, Profit) were of a numeric data type.

Rationale: The numerical columns were required for mathematical operations in scaling and modeling.

```
numerical_features = ['Amount', 'Quantity']
target_variable = 'Profit'
print("Ensuring numerical features are numeric...")
for col in numerical_features + [target_variable]:
    df[col] = pd.to_numeric(df[col], errors='coerce')
```

- One-Hot Encoding

Categorical features (Category, PaymentMode, State) were converted into numerical representation using one-hot encoding to avoid multicollinearity.

Rationale: Models require numerical input; one-hot encoding represents categories as binary features.

```
categorical_features = ['Category', 'PaymentMode', 'State']

categorical_features_in_df = [col for col in categorical_features if col in df_model.columns]

df_model = pd.get_dummies(df_model, columns=categorical_features_in_df, drop_first=True, dtype=float)

print(f"Applied one-hot encoding to: {categorical_features_in_df}")

final_feature_order = list(df_model.drop(columns=target_variable, errors='ignore').columns)

print(final_feature_order)
```

- Handling Missing Values

Any rows containing missing values after the initial steps were dropped. Initial EDA showed a few missing values in key columns, so this had minimal impact but ensured model compatibility.

Rationale: Most standard regression algorithms cannot handle missing input values.

```
df_model.dropna(inplace=True)

print(f"Dropped rows with missing values. Final modeling shape: {df_model.shape}"

X = df_model.drop(columns=target_variable)

y = df_model[target_variable]

print(f"Final shape of features X: {X.shape}")

print(f"Final shape of target y: {y.shape}")
```

- Train/Test Split

The preprocessed data was split into training (80%) and testing (20%) sets using a fixed random_state for reproducibility.

Rationale: Essential for evaluating model performance on unseen data and preventing overfitting.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
    print(f"\nPreprocessed data split into training and testing sets.")
    print(f"Training features shape: {X_train.shape}")
    print(f"Testing features shape:  {X_test.shape}")
    cols_to_scale = [col for col in numerical_features if col in X_train.columns]
    print(f"\nColumns identified for scaling: {cols_to_scale}")
```

- Feature Scaling

Numerical features in the training and test sets were scaled using StandardScaler. The scaler was fit only on the training data and then used to transform both sets.

Rationale: This prevents features with larger ranges from dominating the model (especially important for Linear Regression) and ensures no data leakage from the test set during the scaling process.

```
if cols_to_scale:
        scaler = StandardScaler()
        print("Fitting StandardScaler on training data numerical columns...")
        scaler.fit(X_train[cols_to_scale])
        print("Transforming numerical columns in training and testing data...")
```

```
    X_train_scaled = X_train.copy()
    X_test_scaled = X_test.copy()
    X_train_scaled[cols_to_scale] = scaler.transform(X_train[cols_to_scale])
    X_test_scaled[cols_to_scale] = scaler.transform(X_test[cols_to_scale])
    print("Scaling complete.")
    print("\nScaled Training Data Head (Numerical Columns):")
    print(X_train_scaled[cols_to_scale].head())
else:
    print("\nNo numerical columns found to scale. Using unscaled data.")
    X_train_scaled = X_train.copy()
    X_test_scaled = X_test.copy()
```

4. BUSINESS ANALYTICS QUESTIONS

This project aimed at answering the following business-relevant questions using predictive modeling:

1. How accurately can we predict the profit of a single transaction based on its characteristics like amount, quantity, category, payment mode, and state?

This addresses the core predictive capability, informing stakeholders, for example, Sales and Finance, about the reliability of profit estimates for planning and performance assessment.

2. Which transaction characteristics are the strongest predictors of profit according to our models?

This seeks to identify the key drivers of profitability, providing insights for Marketing, Product, and Strategy teams to focus efforts on high-impact factors.

3. Can our predictive models help identify specific customer segments or transaction types likely to yield significantly higher or lower profit than average?

This focuses on using the model to flag notable transaction profiles, helping Sales and Strategy teams replicate success or investigate underperforming segments.

5. PREDICTIVE MODELING

A supervised machine learning approach using regression was employed to predict the continuous Profit variable. Two models were selected for comparison:

1. Multiple Linear Regression: Chosen as a standard baseline model due to its simplicity and interpretability.
2. Random Forest Regressor: An ensemble method chosen for its ability to capture non-linear relationships and feature interactions, often yielding higher accuracy than linear models.

Process:

Both models were trained on the preprocessed and scaled training data (X_train_scaled, y_train). Their performance was then evaluated on the unseen, preprocessed, and scaled test data (X_test_scaled, y_test) using standard regression metrics.

Evaluation Metrics:

RMSE (Root Mean Squared Error): Measures the average magnitude of the errors, and a lower RMSE is better.

MAE (Mean Absolute Error): Measures the average absolute error. Lower is better, and it's less sensitive to outliers than RMSE.

R-squared (R^2): Represents the proportion of variance in the target variable explained by the model. For R^2, the closer to 1, the better the model.

Results:

	RMSE	MAE	R^2
Multiple Linear Regression	853.3150	657.4731	0.4253
Random Forest Regressor	787.2500	572.0816	0.5109

The Random Forest Regressor demonstrated superior performance compared to the Multiple Linear Regression model across all metrics. It achieved lower prediction errors (RMSE and MAE) and explained a higher proportion of the variance in profit (higher R^2 score). This suggests that the non-linear relationships captured by the Random Forest are important for predicting profit in this dataset.

Multiple Linear Regression was trained as a baseline. While its overall performance (R^2=0.4253) was lower than Random Forest, its coefficients provide direct interpretability of feature relationships (see Appendix, Table A.2 for full details).

The Python code for Model Training, Model Evaluation, and Model Comparison is in the Appendix section.

6. INSIGHTS AND RECOMMENDATIONS

Based on the EDA and modeling results, several key insights and recommendations can be drawn:

Insights:

Moderate Predictability: While the Random Forest model outperformed Linear Regression, its R^2 score of approximately 0.51 indicates it explains about half the variability in profit. This suggests that while the model captures significant trends, other unmeasured factors or inherent randomness also influence profit, and individual predictions will have a degree of uncertainty.

Dominance of Financial Scale: The feature importance analysis clearly shows that Amount is the most critical predictor of profit, followed by Quantity. These two factors heavily outweigh the influence of Category, State, or PaymentMode in the Random Forest model's predictions.

Potential for Segmentation: Although categorical features had lower overall importance in the current model, the EDA showed variations in profit distributions across categories and states. This suggests potential value in segmenting analysis further, perhaps focusing on specific sub-categories or state-category interactions manually or with more complex models.

Variable coefficients: Linear Regression offers high interpretability through its coefficients. Each coefficient represents the estimated average change in Profit for a one-unit increase in the corresponding feature, holding all other features constant. For instance, the model assigned the largest positive coefficient to the scaled Amount feature (753.06), suggesting a strong positive linear relationship with Profit within the model's assumptions. Other features like PaymentMode_Credit Card (355.53) also showed positive coefficients, while State_Ohio showed a negative one (-127.45) relative to the baseline state (the one dropped during encoding). The full list of coefficients can be found in the Appendix (Table A.2). While

interpretable, the lower overall accuracy (R^2) of this model compared to the Random Forest suggests these linear relationships don't fully capture the underlying data complexity.

Recommendations:

Utilize Random Forest for Directional Insights: Use the developed Random Forest model primarily for identifying transactions likely to be high or low profit, rather than relying on it for precise point forecasts, given the moderate R^2. This can help prioritize sales follow-ups or inventory checks.

Focus Strategy on Key Levers: Since Amount and Quantity are the dominant drivers identified by the model, business strategies aimed at increasing profitability should prioritize initiatives that boost average transaction value (e.g., upselling, bundling) or the number of items per transaction.

Feature Engineering: Create new features, such as Profit Margin (Profit / Amount), average item value (Amount / Quantity), or incorporate time-based features if Order Date is reintroduced. Re-evaluate, including Sub-Category.

Deeper Segment Analysis: Conduct targeted analysis outside the current model to understand why certain categories, states, or payment modes might yield different profit levels (as suggested by EDA), even if their overall predictive importance in the RF model was low. This could involve looking at specific product margins or regional costs.

7. ETHICS AND INTERPRETABILITY

Ethical Considerations:

Data Bias: The dataset might contain inherent biases. For example, sales might be concentrated in specific States or Categories, leading the model to perform better for these majority groups and

potentially poorly for underrepresented segments. Using predictions for resource allocation without acknowledging this could unfairly disadvantage certain regions or product lines.

Data Privacy: Customer names were identified and dropped during preprocessing. It's crucial to ensure that no other Personally Identifiable Information (PII) remains and that the data is handled securely according to privacy regulations. Anonymization techniques should be employed if necessary.

Transparency in Use: If the model's predictions influence decisions affecting sales staff for example, commissions and performance reviews, or resource allocation, the model's limitations (moderate accuracy, key drivers) should be communicated transparently to stakeholders. Over-reliance on imperfect predictions should be avoided.

Interpretability:

Linear Regression: This model is highly interpretable. The coefficients associated with each feature directly indicate the predicted change in profit for a one-unit change in that feature, assuming all other features are held constant.

Random Forest Regressor: As an ensemble model, Random Forest is inherently less interpretable than Linear Regression (often considered a "black box"). However, we gained significant insight through:

Feature Importances: This technique provided a ranked list of which features the model relied on most heavily for its predictions. Amount and Quantity were dominant, and this provides a global understanding of drivers. However, feature importance doesn't easily explain how a feature influences the prediction for a single specific transaction, for example, does higher quantity always increase predicted profit, or does it depend on Amount.

APPENDIX

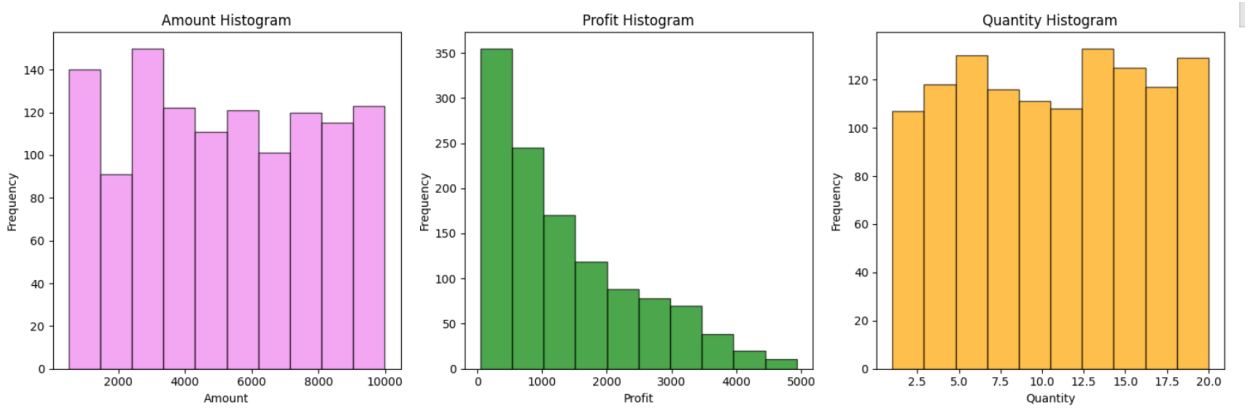


Figure A1: Amount, Profit, and Quantity Histograms

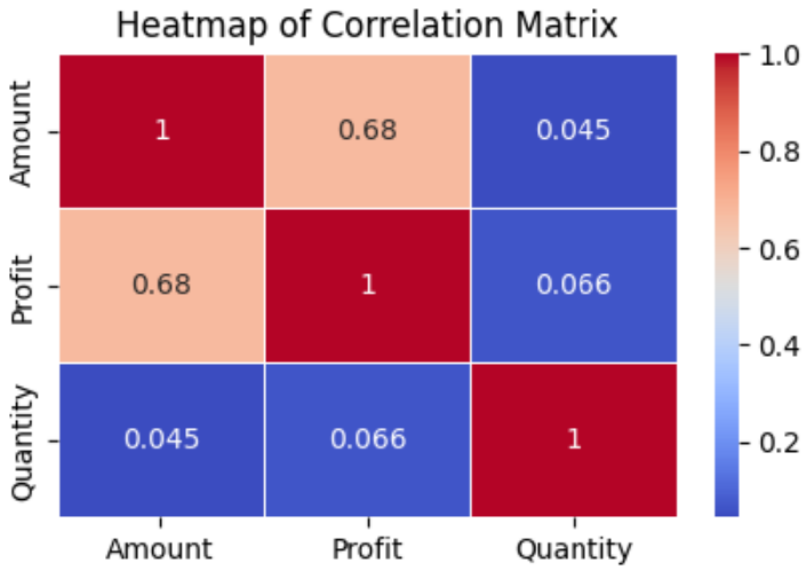


Figure A2: Correlation Matrix

Model Performance Metrics

```
Linear Regression:
  RMSE (Root Mean Squared Error): 853.3150
  MAE (Mean Absolute Error):      657.4731
  R² (R-squared):                 0.4253

Random Forest Regressor:
  RMSE (Root Mean Squared Error): 787.2500
  MAE (Mean Absolute Error):      572.0816
  R² (R-squared):                 0.5109

--- Performance Comparison Table ---
                            RMSE       MAE  R-squared (R²)
Model
Linear Regression        853.315  657.4731          0.4253
Random Forest Regressor  787.250  572.0816          0.5109
```

Table A1: Model Performance Matrix

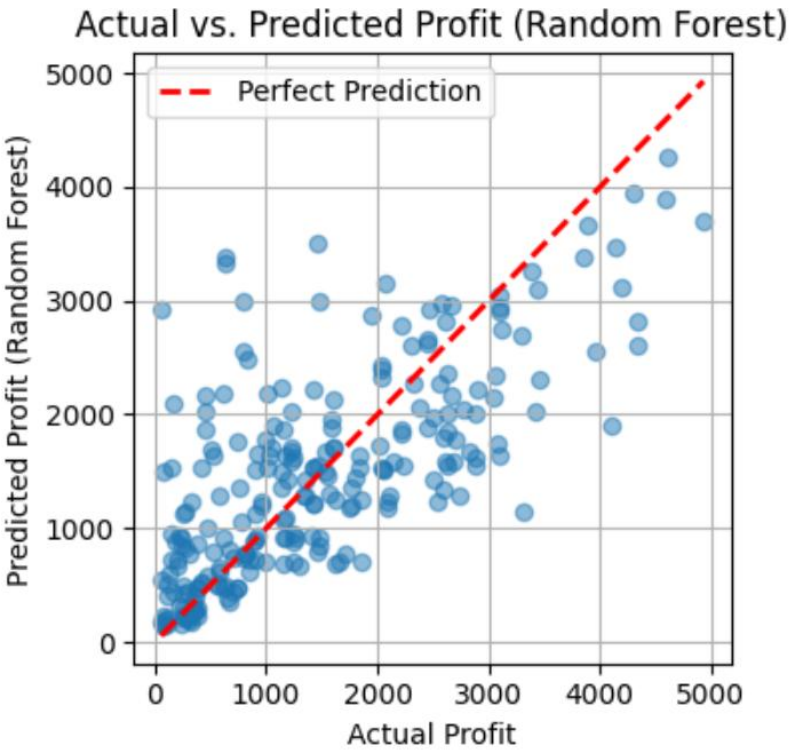


Figure A3: Random Forest Actual vs Predicted

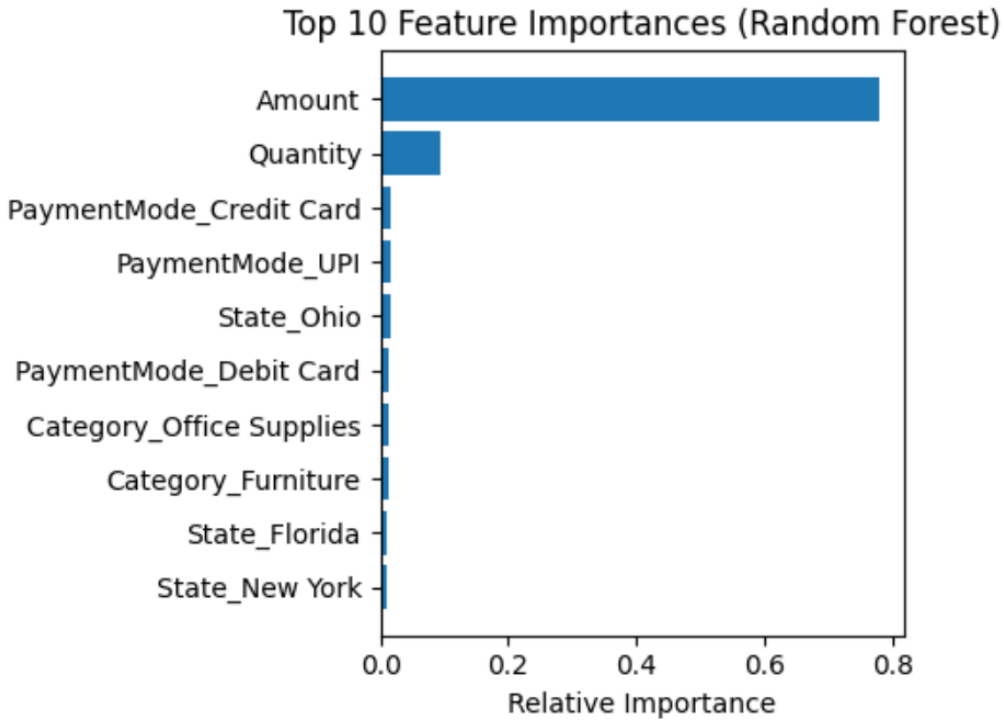


Figure A4: Random Forest Model Features Importance

```
--- Linear Regression Coefficients ---
Intercept: 968.3896

Coefficients (sorted by absolute magnitude):
                          Coefficient
Amount                       753.0556
PaymentMode_Credit Card      355.5338
PaymentMode_EMI              310.1842
PaymentMode_UPI              287.9379
PaymentMode_Debit Card       276.4309
State_Florida                205.5144
State_Ohio                  -127.4507
Category_Furniture           118.4019
Category_Office Supplies      88.8367
State_New York                86.8182
State_Texas                   69.6310
Quantity                      64.6067
State_Illinois               -61.9145
```

Table A2: Linear regression coefficients

Python code

```
import kagglehub

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.linear_model import LinearRegression

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score



file_path = kagglehub.dataset_download('shantanugarg274/sales-dataset', path='Sales Dataset.csv')

df = pd.read_csv(file_path)



print(f"Dataset '{file_path}' loaded successfully.")

print(f"Shape of the dataset: {df.shape}")

print("\nFirst 5 rows:")

print(df.head())

print("\nColumn Info:")

df.info()

print("\nSummary Statistics:")

print(df.describe())



# Create histograms
```

```
plt.figure(figsize=(15, 5))


plt.subplot(1, 3, 1)
df.Amount.plot(kind="hist", bins=10, title='Amount Histogram', color='violet',edgecolor='black', alpha=0.7)
plt.xlabel('Amount')


plt.subplot(1, 3, 2)
df.Profit.plot(kind="hist", bins=10, title='Profit Histogram', color='green',edgecolor='black', alpha=0.7)
plt.xlabel('Profit')


plt.subplot(1, 3, 3)
df.Quantity.plot(kind="hist", bins=10, title='Quantity Histogram', color='orange',edgecolor='black', alpha=0.7)
plt.xlabel('Quantity')


plt.tight_layout()
plt.show()


correlation_matrix = df[['Amount', 'Profit', 'Quantity']].corr()
plt.figure(figsize=(5, 3))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Heatmap of Correlation Matrix')
plt.show()


cols_to_drop = ['Order ID', 'CustomerName', 'City', 'Sub-Category', 'Order Date', 'Year-Month']
categorical_features = ['Category', 'PaymentMode', 'State']
numerical_features = ['Amount', 'Quantity']
```

```
target_variable = 'Profit'


print("\n--- Initial Cleaning & Preprocessing ---")
df.drop(columns=cols_to_drop, inplace=True, errors='ignore')
print(f"Dropped identifier/high-cardinality/unused columns (incl. Sub-Category).")
print("Ensuring numerical features are numeric...")
for col in numerical_features + [target_variable]:
    df[col] = pd.to_numeric(df[col], errors='coerce')


# Selecting only columns needed for modeling
all_feature_cols = [col for col in numerical_features + categorical_features if col in df.columns]
cols_for_model = all_feature_cols + [target_variable]
df_model = df[cols_for_model].copy()
print(f"Selected columns for modeling: {cols_for_model}")


# One-Hot Encode Categorical Features
print("\n--- One-Hot Encoding Categorical Features ---")
categorical_features_in_df = [col for col in categorical_features if col in df_model.columns]
df_model = pd.get_dummies(df_model, columns=categorical_features_in_df, drop_first=True, dtype=float)
print(f"Applied one-hot encoding to: {categorical_features_in_df}")
print("Columns after encoding:")
final_feature_order = list(df_model.drop(columns=target_variable, errors='ignore').columns)
print(final_feature_order)


# Final NaN Drop & Feature/Target Definition
print("\n--- Defining Features (X) and Target (y) ---")
```

```
df_model.dropna(inplace=True)
print(f"Dropped rows with missing values. Final modeling shape: {df_model.shape}")

X = df_model.drop(columns=target_variable)
y = df_model[target_variable]

print(f"Final shape of features X: {X.shape}")
print(f"Final shape of target y: {y.shape}")

if 'X' in locals() and 'y' in locals() and not X.empty:

    print("\n---Splitting & Scaling ---")

    # Split Data into Training and Testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
    print(f"\nPreprocessed data split into training and testing sets.")
    print(f"Training features shape: {X_train.shape}")
    print(f"Testing features shape:  {X_test.shape}")

    cols_to_scale = [col for col in numerical_features if col in X_train.columns]
    print(f"\nColumns identified for scaling: {cols_to_scale}")

    # Scale Numerical Features
    if cols_to_scale:
        scaler = StandardScaler()
```

```
        print("Fitting StandardScaler on training data numerical columns...")
        scaler.fit(X_train[cols_to_scale])

        print("Transforming numerical columns in training and testing data...")
        X_train_scaled = X_train.copy()
        X_test_scaled = X_test.copy()

        X_train_scaled[cols_to_scale] = scaler.transform(X_train[cols_to_scale])
        X_test_scaled[cols_to_scale] = scaler.transform(X_test[cols_to_scale])

        print("Scaling complete.")
        print("\nScaled Training Data Head (Numerical Columns):")
        print(X_train_scaled[cols_to_scale].head())

    else:
        print("\nNo numerical columns found to scale. Using unscaled data.")
        X_train_scaled = X_train.copy()
        X_test_scaled = X_test.copy()

# Model Training
print("\n--- Model Training ---")

# --- Model 1: Multiple Linear Regression ---
print("\nTraining Multiple Linear Regression...")
linear_model = LinearRegression()
```

```
# Train the model on the scaled training data
linear_model.fit(X_train_scaled, y_train)
print("Linear Regression training complete.")

# --- Model 2: Random Forest Regressor ---
print("\nTraining Random Forest Regressor...")
rf_model = RandomForestRegressor(n_estimators=100,
                                 random_state=42,
                                 n_jobs=-1,
                                 max_depth=10,
                                 min_samples_split=10)

# Train the model on the scaled training data
rf_model.fit(X_train_scaled, y_train)
print("Random Forest Regressor training complete.")

print("\n--- Model Evaluation ---")

# Make predictions on the test set
print("Making predictions on the test set...")
y_pred_lr = linear_model.predict(X_test_scaled)
y_pred_rf = rf_model.predict(X_test_scaled)
print("Predictions complete.")

# Evaluate Linear Regression
print("\nEvaluating Linear Regression...")
```

```
rmse_lr = np.sqrt(mean_squared_error(y_test, y_pred_lr))
mae_lr = mean_absolute_error(y_test, y_pred_lr)
r2_lr = r2_score(y_test, y_pred_lr)

# Evaluate Random Forest
print("Evaluating Random Forest Regressor...")
rmse_rf = np.sqrt(mean_squared_error(y_test, y_pred_rf))
mae_rf = mean_absolute_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)

# Organize and Print Results
print("\n--- Model Performance Metrics ---")
print("\nLinear Regression:")
print(f"  RMSE (Root Mean Squared Error): {rmse_lr:.4f}")
print(f"  MAE (Mean Absolute Error):      {mae_lr:.4f}")
print(f"  R² (R-squared):                 {r2_lr:.4f}")

print("\nRandom Forest Regressor:")
print(f"  RMSE (Root Mean Squared Error): {rmse_rf:.4f}")
print(f"  MAE (Mean Absolute Error):      {mae_rf:.4f}")
print(f"  R² (R-squared):                 {r2_rf:.4f}")

# Create a comparison table
results_df = pd.DataFrame({
    'Model': ['Linear Regression', 'Random Forest Regressor'],
    'RMSE': [rmse_lr, rmse_rf],
```

```
    'MAE': [mae_lr, mae_rf],
    'R-squared (R²)': [r2_lr, r2_rf]
  })
# Set Model as index for better readability
results_df.set_index('Model', inplace=True)


print("\n--- Performance Comparison Table ---")
print(results_df.round(4))


# Model Comparison Summary
print("\n--- Model Comparison Summary ---")
print(f"Comparing the two models based on test set performance:")
print(results_df.round(4))


# Determine the better model based on R-squared
better_model_name = results_df['R-squared (R²)'].idxmax()
print(f"\nConclusion: The {better_model_name} performed better.")
print(f"- It achieved a lower RMSE ({results_df.loc[better_model_name, 'RMSE']:.4f} vs {results_df.drop(better_model_name)['RMSE'].values[0]:.4f})")
print(f"- It achieved a lower MAE ({results_df.loc[better_model_name, 'MAE']:.4f} vs {results_df.drop(better_model_name)['MAE'].values[0]:.4f})")
print(f"- It explained more variance in Profit (R² = {results_df.loc[better_model_name, 'R-squared (R²)']:.4f} vs {results_df.drop(better_model_name)['R-squared (R²)'].values[0]:.4f})")
print("\nThis suggests the Random Forest's ability to capture non-linear relationships was beneficial for this dataset.")
```

```
# Visualization: Actual vs. Predicted Plot for Random Forest
print("\nGenerating Actual vs. Predicted plot for Random Forest...")
plt.figure(figsize=(4, 4))
plt.scatter(y_test, y_pred_rf, alpha=0.5)
# Add a line for perfect predictions (y_test = y_pred)
min_val = min(y_test.min(), y_pred_rf.min())
max_val = max(y_test.max(), y_pred_rf.max())
plt.plot([min_val, max_val], [min_val, max_val], '--r', linewidth=2, label='Perfect Prediction')
plt.xlabel("Actual Profit")
plt.ylabel("Predicted Profit (Random Forest)")
plt.title("Actual vs. Predicted Profit (Random Forest)")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()


# Visualization: Feature Importance for Random Forest
print("\nGenerating Feature Importance plot for Random Forest...")


if 'final_feature_order' in locals() and final_feature_order:
    feature_names = final_feature_order
    importances = rf_model.feature_importances_
    indices = np.argsort(importances)[::-1]


    top_n = 10
    indices = indices[:top_n]
```

```
        plt.figure(figsize=(5, 4))
        plt.title(f"Top {top_n} Feature Importances (Random Forest)")
        plt.barh(range(len(indices)), importances[indices], align='center')
        plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
        plt.xlabel('Relative Importance')
        plt.gca().invert_yaxis()
        plt.tight_layout()
        plt.show()

else:
        print("Could not generate feature importance plot: Feature names list ('final_feature_order') not found or empty.")
print("\n--- Linear Regression Coefficients ---")

    intercept = linear_model.intercept_
    print(f"Intercept: {intercept:.4f}\n")

    coefficients = linear_model.coef_

    feature_names = X_train_scaled.columns

    coeff_df = pd.DataFrame(coefficients, index=feature_names, columns=['Coefficient'])

    coeff_df['Abs_Coefficient'] = coeff_df['Coefficient'].abs()
    coeff_df = coeff_df.sort_values(by='Abs_Coefficient', ascending=False)

    print("Coefficients (sorted by absolute magnitude):")
    print(coeff_df[['Coefficient']].round(4))
```