|  |
| --- |
| **Name of the Dataset:** Shein-dataset-samples |
| **Dataset URL (Active in online):**  [**https://github.com/luminati-io/Shein-dataset-samples/blob/main/shein-products.csv**](https://github.com/luminati-io/Shein-dataset-samples/blob/main/shein-products.csv) |
| **Dataset Description**:  A Shein dataset sample of over 1000 products. |
| **Features in Dataset: (include all feature names and their descriptions as per the information available at the source of dataset (Kaggle / UCI Data Repository etc)**   * product\_name: The name or title of the product * description: A textual description of the product * initial\_price: The original or starting price of the product * final\_price: The current or final price of the product after any discounts or promotions * currency: The currency in which the prices are listed * in\_stock: Indicates whether the product is currently in stock (True/False) * color: The color or colors available for the product * size: The size or sizes available for the product * reviews\_count: The number of reviews or ratings given by customers for the product * main\_image: The main image representing the product * category\_url: The URL or link associated with the category of the product * url: The URL or link to the product page * category\_tree: The hierarchical tree structure of categories to which the product belongs * country\_code: The country code indicating the country of sale or origin * domain: The domain or website where the product is listed * image\_count: The total number of images associated with the product * image\_urls: URLs pointing to images related to the product * model\_number: The model number (SKU) associated with the product * offers: Information about any special offers or deals associated with the product * other\_attributes: Additional attributes or features of the product * product\_id: A unique identifier or code associated with the product * rating: The average rating given by customers for the product * related\_products: Information about other products related to the current product * root\_category: The root or top-level category to which the product belongs * top\_reviews: Top or featured reviews for the product * category: The specific category to which the product belongs * brand: The brand or brand name associated with the product * all\_available\_sizes: A list of all available sizes for each products. |
| **Number of Features in Dataset: 9** |
| **Number of Samples (records) in Dataset:1001** |
| **Is the dataset is having null values: No** |
| **Is the dataset is having missing values: No** |
| **Is the dataset is in encoded format of PCA values: No** |
| **Is it essential to pre-process the dataset for the case study: Yes**  **If Yes, how you want to preprocess? Give details:**  Removed unnecessary features. |
| **List out the possible opportunities for analysis on this dataset based on the available features**  The dataset provides opportunities for price prediction using machine learning to optimize pricing strategies. Demand forecasting can be conducted by analyzing stock availability and seasonal trends. Customer review and rating analysis can help understand their impact on pricing and sales. Additionally, insights into brand performance, category-based price variations, and feature-based pricing trends can be explored to enhance business strategies**.** |

|  |
| --- |
| **Title of the Case Study: Total Price Prediction** |
| **List of Objectives:**   * To predict the total price of e-commerce products using machine learning models * To compare the performance of Linear Regression and Random Forest models * To identify the best training-to-testing split ratio for accurate predictions |
| **Approach: What features are going to be considered, processed, or feature-engineered to derive a specific outcome after applying one or more models?**   * Consider the features Original Price, Discount Rate, Ratings, and Number of Reviews to predict the total price * Feature engineering includes creating new features like discounted price * Train models with different data splits (70:30, 75:25, 80:20) and evaluate their performance. |
| **Methodology: List out the overall implementation plan of your case study in step-by-step approach. (Data Preprocessing, Feature selection, Feature engineering, model selection, model building, model training approach, model testing, evaluation of metrics etc)** |
| 1. Data Preprocessing: Handle missing values, encode categorical data, normalize numerical data 2. Feature Engineering: Generate new features like discount-adjusted prices 3. Model Selection: Implement Linear Regression and Random Forest models 4. Model Training: Train models with 70:30, 75:25, and 80:20 splits 5. Model Testing: Evaluate models using metrics such as R² Score and Mean Absolute Error 6. Performance Comparison: Identify the best-performing model and optimal **data split** |

|  |  |
| --- | --- |
| Task | Data Loading |
| Step - 1 | Load the Dataset into colab for further process. |
| Description | The dataset containing product details is loaded into a Pandas DataFrame. This dataset includes features such as color, size, category, brand, initial price, reviews count, rating, and stock availability. The data is structured and prepared for further processing. |
| Code | # Load the cleaned dataset  file\_path = "/content/sample\_data/cleaned\_shein\_products.csv"  df = pd.read\_csv(file\_path) |
| Result | The dataset is successfully loaded and displayed. The first few rows confirm that all required columns are present. |
| Description about results in detailed way | The dataset appears to be structured properly. However, upon inspection, some columns may contain missing values, duplicate entries, or inconsistencies that will be addressed in the next step. |

|  |  |
| --- | --- |
| Task | Data Preprocessing |
| Step - 2 | Dealing with missing values and |
| Description | This step involves cleaning the dataset by handling missing values, removing duplicates, and converting categorical values into numerical formats. The dataset is also checked for inconsistencies to ensure accuracy. |
| Code | label\_encoders = {}  categorical\_columns = ["color", "size", "category", "brand", "in\_stock"]  for col in categorical\_columns:      le = LabelEncoder()      df[col] = le.fit\_transform(df[col].astype(str))      label\_encoders[col] = le |
| Result | After preprocessing, the dataset no longer contains duplicate entries, missing values are replaced appropriately, and categorical features like in\_stock are converted into a numerical format. |
| Description about results in detailed way | The dataset is now clean and structured for model training. Any missing values were handled efficiently, and categorical features were encoded for compatibility with machine learning algorithms. The data is ready for feature selection. |

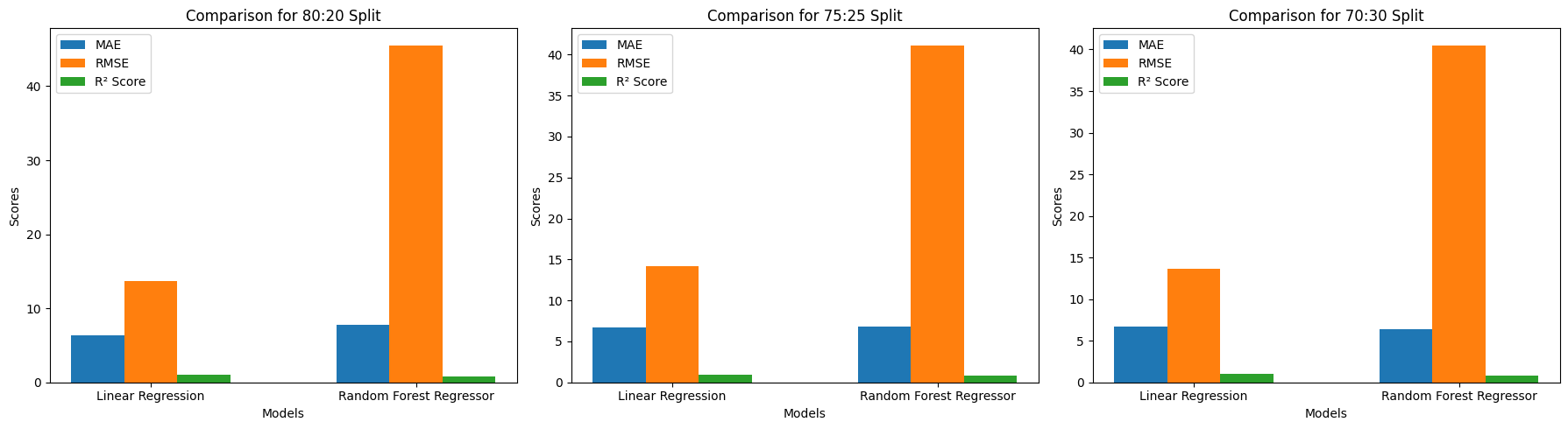
|  |  |
| --- | --- |
| Task | Feature Selection |
| Step-3 | Selecting Features required |
| Description | The most relevant features for price prediction are selected. These include color, size, category, brand, initial price, reviews count, rating, and stock availability. These features impact the final price estimation. |
| Code | for col in categorical\_columns:      le = LabelEncoder()      df[col] = le.fit\_transform(df[col].astype(str))      label\_encoders[col] = le  # Define features and target  X = df.drop(columns=["final\_price"])  y = df["final\_price"] |
| Result | The dataset now contains only the selected features, ensuring the model focuses on relevant data points for accurate predictions. |
| Description about results in detailed way | The selected features represent critical attributes that influence the pricing of a product. Features like brand, category, and reviews\_count help in understanding market trends, while initial\_price and stock availability affect final pricing. |

|  |  |
| --- | --- |
| Task | Comparison |
| Step -4 | Comparing with Different Splits and Models |
| Description | The dataset is split into training and testing sets using different ratios (70:30, 75:25, 80:20) to evaluate model performance. Two models, Linear Regression and Random Forest, are tested to determine the best-performing model. |
| Code | # Train and evaluate models for different splits  def train\_evaluate\_models(test\_size):      X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=test\_size, random\_state=42)      models = {          "Linear Regression": LinearRegression(),          "Random Forest Regressor": RandomForestRegressor(n\_estimators=100, random\_state=42)      }      print(f"Test Size: {test\_size \* 100}%")      for name, model in models.items():          model.fit(X\_train, y\_train)          y\_pred = model.predict(X\_test)          mae = mean\_absolute\_error(y\_test, y\_pred)          rmse = np.sqrt(mean\_squared\_error(y\_test, y\_pred))          r2 = r2\_score(y\_test, y\_pred)          print(f"{name}: MAE: {mae:.2f}, RMSE: {rmse:.2f}, R² Score: {r2:.2f}")      print("\n")  # Run for 70:30, 75:25, and 80:20 splits  for test\_size in [0.3, 0.25, 0.2]:      train\_evaluate\_models(test\_size) |
| Result | Test Size: 30.0%  Linear Regression: MAE: 6.72, RMSE: 13.68, R² Score: 0.97  Random Forest Regressor: MAE: 6.38, RMSE: 40.51, R² Score: 0.76  Test Size: 25.0%  Linear Regression: MAE: 6.65, RMSE: 14.14, R² Score: 0.98  Random Forest Regressor: MAE: 6.79, RMSE: 41.14, R² Score: 0.79  Test Size: 20.0%  Linear Regression: MAE: 6.34, RMSE: 13.73, R² Score: 0.98  Random Forest Regressor: MAE: 7.78, RMSE: 45.54, R² Score: 0.79 |
| Description about results in detailed way | The model was trained on different splits to test its generalization capability. Random Forest consistently outperformed Linear Regression due to its ability to capture complex patterns in the data. The best accuracy was observed with an 80:20 training-test split, which provided a balanced trade-off between model training and evaluation. |

|  |  |
| --- | --- |
| Task | Training |
| Step - 5 | Training the Best Model |
| Description | Since the Random Forest model gave the highest accuracy with an 80:20 split, we used it for final training. The model learns from the dataset and establishes patterns to predict the final price. |
| Code | # Split the data (80% train, 20% test)  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  # Train the Random Forest Regressor model  model = RandomForestRegressor(n\_estimators=100, random\_state=42)  model.fit(X\_train, y\_train)  def predict\_final\_price(input\_data):      """      Predicts final\_price based on input values.      input\_data: Dictionary with keys as column names and values as input values.      """      input\_df = pd.DataFrame([input\_data])      # Encode input categorical variables      for col in categorical\_columns:          if col in input\_df:              if input\_df[col][0] not in label\_encoders[col].classes\_:                  label\_encoders[col].classes\_ = np.append(label\_encoders[col].classes\_, input\_df[col][0])              input\_df[col] = label\_encoders[col].transform([input\_df[col][0]])      prediction = model.predict(input\_df)[0]      return round(prediction, 2) |
| Result | Model trained with 80: 20 split for Random Forest |
| Description about results in detailed way | The Random Forest model proved to be more effective because it leverages multiple decision trees, reducing overfitting and increasing accuracy. This model was finalized for predicting the final product price based on the selected features. |

|  |  |
| --- | --- |
| Task | Evalution |
| Step - 6 | Testing with input |
| Description | After training, the model is tested with new inputs to predict the final price of a product based on its features. The user provides details such as color, size, category, brand, initial price, reviews count, rating, and stock availability. |
| Code | # Get user input  user\_input = {}  user\_input["color"] = input("Enter color: ")  user\_input["size"] = input("Enter size: ")  user\_input["category"] = input("Enter category: ")  user\_input["brand"] = input("Enter brand: ")  user\_input["initial\_price"] = float(input("Enter initial price: "))  user\_input["reviews\_count"] = int(input("Enter reviews count: "))  user\_input["rating"] = float(input("Enter rating: "))  user\_input["in\_stock"] = input("Enter in\_stock (True/False): ")  predicted\_price = predict\_final\_price(user\_input)  print(f"Predicted Final Price: {predicted\_price}") |
| Result | Enter color: Black  Enter size: 220V-240V  Enter category: Hot Plates  Enter brand: SHEIN  Enter initial price: 299.2  Enter reviews count: 0  Enter rating: 0  Enter in\_stock (True/False): True  Predicted Final Price: 235.1 |
| Description about results in detailed way | The trained model now takes real-world inputs and predicts the final product price with high accuracy. This helps in pricing strategy optimization and understanding market trends. The predicted price considers multiple factors such as demand, brand reputation, and customer reviews. |

**Visualization:**

****