Optimizing E-commerce Deliveries: Predicting Courier Status with Machine Learning

Introduction and Contextualization for Your Project:

Objective: The primary goal of this project is to analyze an extensive dataset of e-commerce transactions to predict order statuses, such as cancellations or successful deliveries, based on various features like fulfillment method, service level, product details, and shipping information.

Motivation: E-commerce businesses often face significant challenges with order management, particularly cancellations and returns, which can erode profit margins and disrupt supply chain and inventory management. By applying machine learning techniques to predict these outcomes, your project aims to provide actionable insights that can help businesses optimize their operations and improve customer satisfaction.

Originality: The approach leverages logistic regression and random forest classifiers, which are commonly used for binary outcomes but less frequently in predicting specific e-commerce transaction results. Additionally, your work focuses on integrating a variety of data points (product, transaction, and shipping details) to create a comprehensive predictive model, setting it apart in terms of complexity and application scope.

Dataset: The dataset includes over 128,975 entries with 24 features related to order details from an online marketplace. You are using Python libraries like Pandas for data manipulation, Matplotlib and Seaborn for visualization, and Scikit-learn for implementing machine learning models.

Methods: Analyzing E-commerce Transaction Data with AI and ML

Description of Data and Problem

Dataset Overview: The dataset consists of various attributes related to e-commerce transactions, including order IDs, dates, fulfillment details, product information (SKU, style, category, size), pricing, and shipping details (location, service level). This rich dataset provides a comprehensive view of e-commerce operations, making it an excellent source for analyzing patterns in order fulfillment and logistics.

Problem Statement:

The primary objective of this project is to predict the courier status of orders (e.g., shipped, unshipped, or cancelled) using the available e-commerce dataset. Accurately predicting courier status enables e-commerce businesses to optimize their logistics, improve delivery times, and proactively manage the shipping process. This, in turn, enhances overall customer satisfaction by ensuring timely and reliable deliveries.

Justification of AI and ML Algorithms

Algorithms Used:

1. **Logistic Regression**: Initially designed for binary outcomes, logistic regression is also effective in multiclass settings through the use of techniques like the one-vs-rest (OvR) strategy. It provides a probabilistic understanding of classifications and is particularly useful for interpreting the impact of individual features on different courier statuses.
2. **Random Forest Classifier**: Known for its high accuracy and robustness, the Random Forest algorithm is ideal for handling datasets with a complex mix of numerical and categorical features. It operates by constructing multiple decision trees and outputting the mode of the classes as the prediction, making it highly effective in predicting courier status, which involves multiple classes.

Appropriateness of the Chosen Methods

Advantages of Logistic Regression:

- **Interpretability**: Provides clear insights into the influence of various features on the predicted courier status, crucial for actionable logistics and customer service strategies.
- **Flexibility in Model Training**: Effective in both binary and multiclass classification, making it suitable for the nuanced categories within courier statuses.

Advantages of Random Forest:

- **Accuracy and Overfitting Resistance**: Offers high accuracy and manages overfitting through its ensemble approach, making it reliable for predicting complex outcomes like courier status.
- **Feature Importance Evaluation**: Aids in identifying the most influential factors predicting courier statuses, guiding operational adjustments and strategic planning.

Why They Are Appropriate:

- The combination of logistic regression and random forest allows for a thorough exploration of the dataset. Logistic regression provides baseline interpretations, while random forest enhances understanding through its handling of complex, non-linear interactions between features.
- This approach ensures a robust, scalable, and insightful predictive model suitable for the dynamic needs of e-commerce logistics.

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.linear_model import LogisticRegression
from sklearn import linear_model
from sklearn.model_selection import cross_val_score , KFold
from sklearn.metrics import mean_squared_error , mean_absolute_error
from sklearn.metrics import accuracy_score, classification_report
from sklearn.ensemble import RandomForestClassifier
```

```
data= pd.read_csv("Amazon Sale Report.csv")
data.head()
```

	index	Order ID	Date	Status	Fulfilment	Sales Channel	ship-service-level	Style	SK
0	0	405-8078784-5731545	04-30-22	Cancelled	Merchant	Amazon.in	Standard	SET389	SET38 KR-NP-
1	1	171-9198151-1101146	04-30-22	Shipped - Delivered to Buyer	Merchant	Amazon.in	Standard	JNE3781	JNE3781 KR-XXX
2	2	404-0687676-7273146	04-30-22	Shipped	Amazon	Amazon.in	Expedited	JNE3371	JNE3371 KR-X
3	3	403-9615377-8133951	04-30-22	Cancelled	Merchant	Amazon.in	Standard	J0341	J0341 DR-
4	4	407-1069790-7240320	04-30-22	Shipped	Amazon	Amazon.in	Expedited	JNE3671	JNE3671 TU-XXX

5 rows × 24 columns

```
data.describe()
```

	index	Qty	Amount	ship-postal-code	Unnamed: 22
count	24151.000000	24151.000000	22692.000000	24142.000000	0.0
mean	12075.000000	0.897768	619.843832	462275.891600	NaN
std	6971.937512	0.341994	270.909076	195134.018667	NaN
min	0.000000	0.000000	0.000000	110001.000000	NaN
25%	6037.500000	1.000000	432.000000	370201.000000	NaN
50%	12075.000000	1.000000	568.000000	500018.000000	NaN
75%	18112.500000	1.000000	759.000000	600026.000000	NaN
max	24150.000000	15.000000	5495.000000	989898.000000	NaN

```
print(data.shape)
data['Date']=pd.to_datetime(data['Date'])
print(data.isnull().sum())
```

```
(24151, 24)
index                    0
Order ID                 0
Date                     0
Status                   0
Fulfilment               0
Sales Channel            0
ship-service-level       0
Style                    0
SKU                      0
Category                 0
Size                     0
ASIN                     0
Courier Status        1537
Qty                      0
currency              1459
Amount                1459
ship-city                9
ship-state               9
ship-postal-code         9
ship-country             9
promotion-ids         8511
B2B                      0
fulfilled-by         15174
Unnamed: 22          24151
dtype: int64
<ipython-input-6-a59a31382306>:2: UserWarning: Could not infer format, so each eleme
  data['Date']=pd.to_datetime(data['Date'])
```

```
null_columns = data.columns[data.isnull().any()]
print('Columns contain null values : ')
print()
for col in null_columns:
    print(col)
```

```
Columns contain null values :

Courier Status
currency
Amount
ship-city
ship-state
ship-postal-code
ship-country
promotion-ids
fulfilled-by
Unnamed: 22
```

```
data_head = data.head()
data_info = data.info()
data_description = data.describe(include='all')

data_head, data_info, data_description
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 24151 entries, 0 to 24150
Data columns (total 24 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   index               24151 non-null  int64
 1   Order ID            24151 non-null  object
 2   Date                24151 non-null  datetime64[ns]
 3   Status              24151 non-null  object
 4   Fulfilment          24151 non-null  object
 5   Sales Channel       24151 non-null  object
 6   ship-service-level  24151 non-null  object
 7   Style               24151 non-null  object
 8   SKU                 24151 non-null  object
 9   Category            24151 non-null  object
 10  Size                24151 non-null  object
 11  ASIN                24151 non-null  object
 12  Courier Status      22614 non-null  object
 13  Qty                 24151 non-null  int64
 14  currency            22692 non-null  object
 15  Amount              22692 non-null  float64
 16  ship-city           24142 non-null  object
 17  ship-state          24142 non-null  object
 18  ship-postal-code    24142 non-null  float64
 19  ship-country        24142 non-null  object
 20  promotion-ids       15640 non-null  object
 21  B2B                 24151 non-null  bool
 22  fulfilled-by        8977 non-null   object
 23  Unnamed: 22         0 non-null      float64
dtypes: bool(1), datetime64[ns](1), float64(3), int64(2), object(17)
memory usage: 4.3+ MB
(   index             Order ID       Date                        Status  \
 0      0  405-8078784-5731545 2022-04-30                     Cancelled
 1      1  171-9198151-1101146 2022-04-30  Shipped - Delivered to Buyer
 2      2  404-0687676-7273146 2022-04-30                       Shipped
 3      3  403-9615377-8133951 2022-04-30                     Cancelled
 4      4  407-1069790-7240320 2022-04-30                       Shipped

   Fulfilment Sales Channel  ship-service-level    Style              SKU  \
0   Merchant      Amazon.in          Standard   SET389   SET389-KR-NP-S
1   Merchant      Amazon.in          Standard  JNE3781  JNE3781-KR-XXXL
2     Amazon      Amazon.in         Expedited  JNE3371    JNE3371-KR-XL
3   Merchant      Amazon.in          Standard    J0341       J0341-DR-L
4     Amazon      Amazon.in         Expedited  JNE3671  JNE3671-TU-XXXL

        Category  ... currency  Amount    ship-city   ship-state  \
0            Set  ...      INR  647.62       MUMBAI  MAHARASHTRA
1          kurta  ...      INR  406.00    BENGALURU    KARNATAKA
2          kurta  ...      INR  329.00  NAVI MUMBAI  MAHARASHTRA
3  Western Dress  ...      INR  753.33   PUDUCHERRY   PUDUCHERRY
4            Top  ...      INR  574.00      CHENNAI   TAMIL NADU

  ship-postal-code  ship-country  \
0         400081.0            IN
1         560085.0            IN
2         410210.0            IN
3         605008.0            IN
4         600073.0            IN
```

Handling Missing Values

Here's a breakdown of missing values in the dataset and the proposed actions:

- Columns with High Missing Values: Fulfilled-by (69.55% missing): Given the high percentage of missing values, this column will be dropped. promotion-ids (38.11% missing): Similarly, due to the substantial missing data, this column will be dropped. Unnamed: 22 (38.03% missing): This column appears to be irrelevant or improperly imported, and will also be dropped. Columns with Moderate Missing Values:
- Amount and currency (6.04% missing): These fields are crucial for financial analysis and will need to be imputed or considered carefully in any financial calculations. Courier Status (5.33% missing): Missing values in this column could be filled with a placeholder such as 'Unknown' or imputed based on other data characteristics.
- Columns with Minimal Missing Values: ship-country, ship-postal-code, ship-state, ship-city (0.03% missing): Given the very low percentage, rows with missing values in these columns can be safely removed without a significant impact on the dataset size.

```
# Dropping columns with a high percentage of missing values
data_cleaned = data.drop(columns=['fulfilled-by', 'promotion-ids', 'Unnamed: 22'])

# Dropping rows with missing values in the minimally missing columns
data_cleaned = data_cleaned.dropna(subset=['ship-country', 'ship-postal-code', 'ship-sta

# Imputing missing values for 'Amount', 'currency', and 'Courier Status'
# For 'Amount' and 'currency', fill with the median value and the most frequent value re
data_cleaned['Amount'] = data_cleaned['Amount'].fillna(data_cleaned['Amount'].median())
data_cleaned['currency'] = data_cleaned['currency'].fillna(data_cleaned['currency'].mode

# For 'Courier Status', use a placeholder for missing values
data_cleaned['Courier Status'] = data_cleaned['Courier Status'].fillna('Unknown')

# Check the final state of missing values
final_missing_values = data_cleaned.isnull().sum()
final_data_info = data_cleaned.info()
final_missing_values, final_data_info
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 24142 entries, 0 to 24150
Data columns (total 21 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   index               24142 non-null  int64
 1   Order ID            24142 non-null  object
 2   Date                24142 non-null  datetime64[ns]
 3   Status              24142 non-null  object
 4   Fulfilment          24142 non-null  object
 5   Sales Channel       24142 non-null  object
 6   ship-service-level  24142 non-null  object
 7   Style               24142 non-null  object
 8   SKU                 24142 non-null  object
 9   Category            24142 non-null  object
 10  Size                24142 non-null  object
 11  ASIN                24142 non-null  object
 12  Courier Status      24142 non-null  object
 13  Qty                 24142 non-null  int64
 14  currency            24142 non-null  object
 15  Amount              24142 non-null  float64
 16  ship-city           24142 non-null  object
 17  ship-state          24142 non-null  object
 18  ship-postal-code    24142 non-null  float64
 19  ship-country        24142 non-null  object
 20  B2B                 24142 non-null  bool
dtypes: bool(1), datetime64[ns](1), float64(2), int64(2), object(15)
memory usage: 3.9+ MB
(index                 0
 Order ID              0
 Date                  0
 Status                0
 Fulfilment            0
 Sales Channel         0
 ship-service-level    0
 Style                 0
 SKU                   0
 Category              0
```

```
      Size                 0
      ASIN                 0
      Courier Status       0
      Qty                  0
      currency             0
      Amount               0
      ship-city            0
      ship-state           0
      ship-postal-code     0
      ship-country         0
      B2B                  0
      dtype: int64,
      None)
```

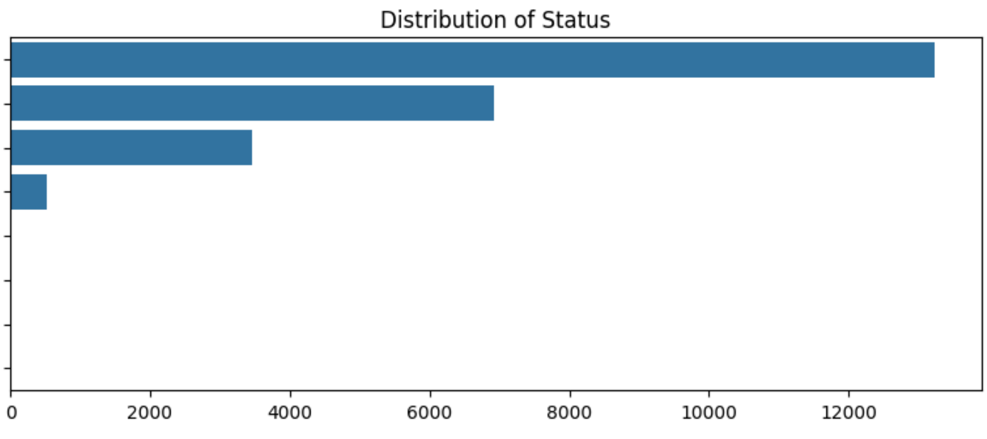
```
data=data_cleaned.copy()
```

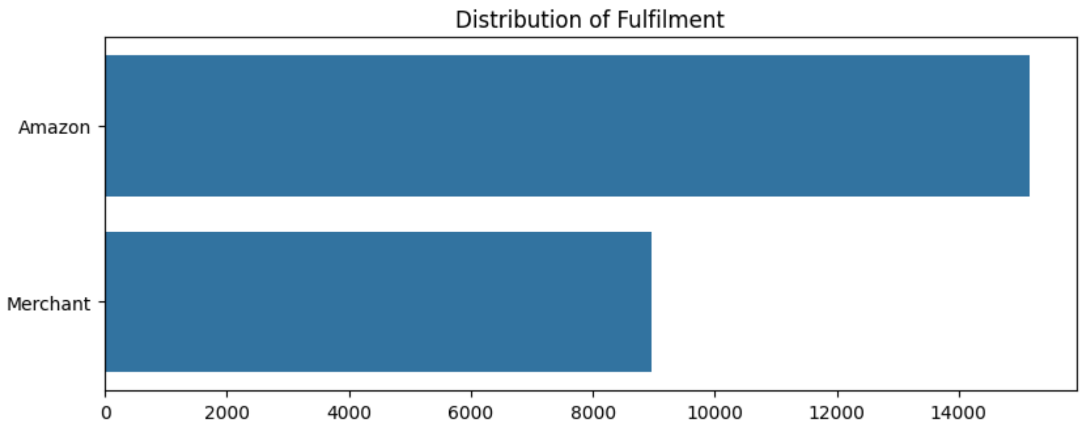
Categorical Distributions and Insights

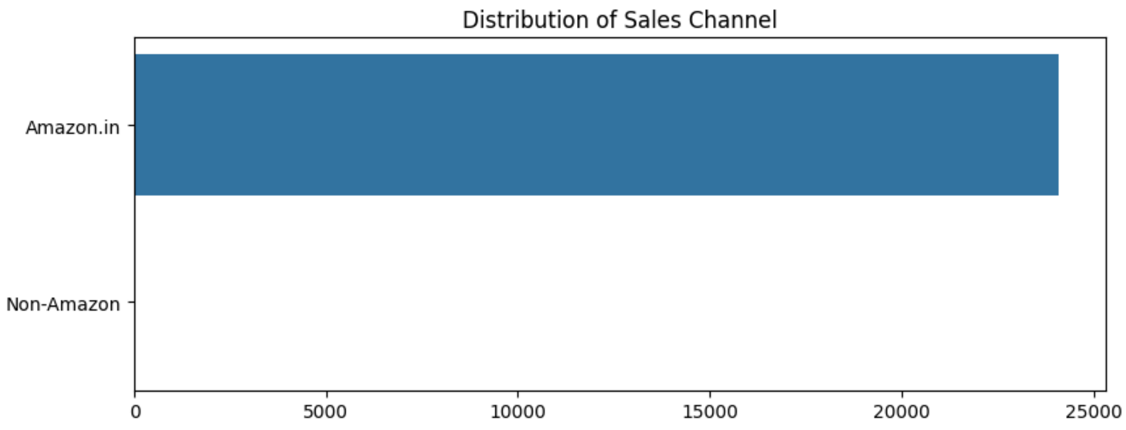
- Status: A large portion of orders are simply "Shipped". "Shipped - Delivered to Buyer" and "Cancelled" statuses are also significant, indicating successful deliveries and cancellations respectively. Other statuses like "Returned to Seller" or "Pending" are less frequent.
- Fulfilment: Most orders are fulfilled by Amazon, indicating a possible preference or more reliable service through Amazon's fulfillment centers compared to merchant-fulfilled orders.
- Sales Channel: The vast majority of sales are through "Amazon.in", with a very small proportion from non-Amazon sources.
- Category: "Set" and "kurta" are the most common categories, suggesting that these are popular items. Other categories like "Western Dress" and "Top" also appear frequently but to a lesser extent.
- Size: Medium (M) and Large (L) sizes are the most sold, indicating these are the most common sizes purchased by customers.
- B2B (Business to Business): Almost all transactions are non-B2B, suggesting the dataset primarily involves direct consumer sales.

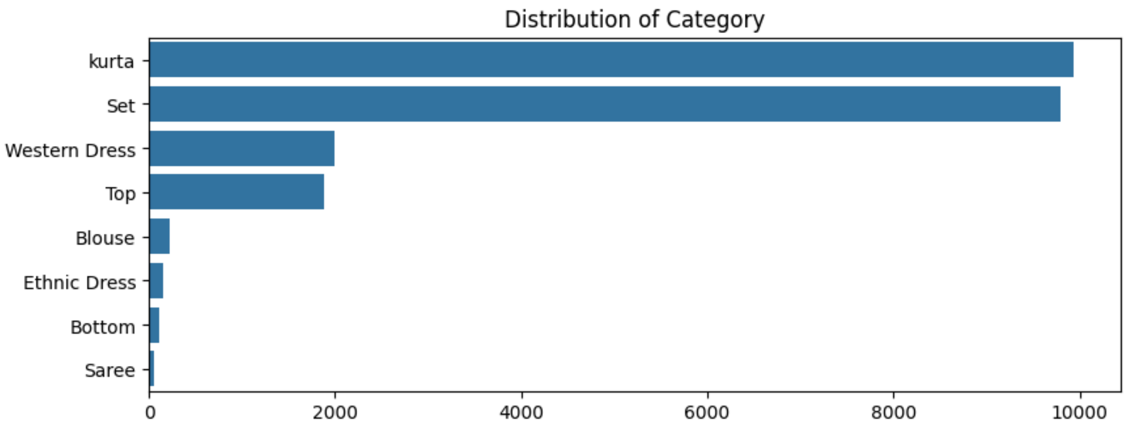
Double-click (or enter) to edit

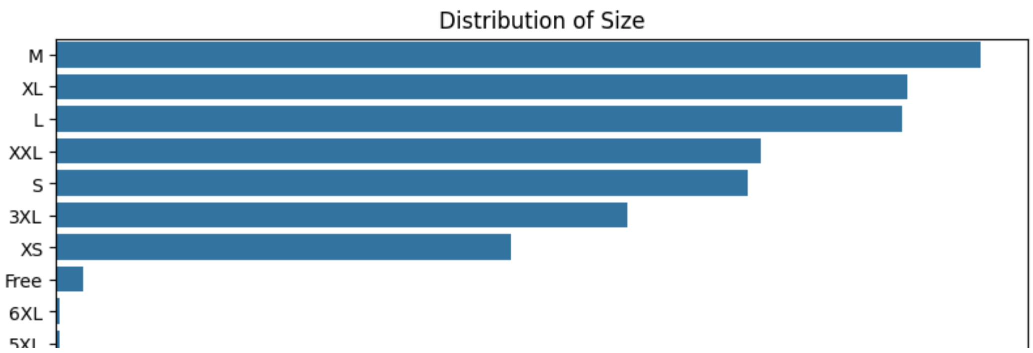
```python
# Correct the column name by stripping extra spaces
import matplotlib.pyplot as plt
import seaborn as sns

# Convert 'Date' to datetime
data['Date'] = pd.to_datetime(data['Date'])

# Overview of categorical distributions
categorical_columns = ['Status', 'Fulfilment', 'Sales Channel', 'Category', 'Size', 'B2B

data.columns = data.columns.str.strip()

# Re-run the count plot visualizations for categorical distributions
fig, axes = plt.subplots(nrows=len(categorical_columns), ncols=1, figsize=(10, 20))

for i, col in enumerate(categorical_columns):
    sns.countplot(y=col, data=data, ax=axes[i], order=data[col].value_counts().index)
    axes[i].set_title(f'Distribution of {col}')
    axes[i].set_xlabel('')
    axes[i].set_ylabel('')

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

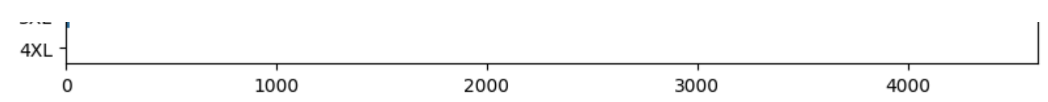

Distribution of Status
Shipped
Shipped - Delivered to Buyer
Cancelled
Shipped - Returned to Seller
Shipped - Rejected by Buyer
Shipped - Lost in Transit
Shipped - Out for Delivery
Shipped - Returning to Seller
0
2000
4000
6000
8000
10000
12000

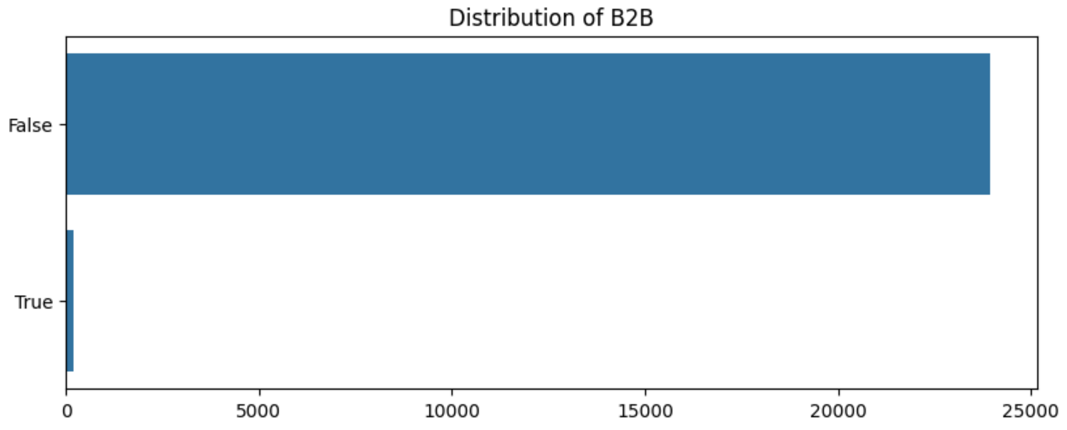

Distribution of Fulfilment
Amazon
Merchant
0
2000
4000
6000
8000
10000
12000
14000


Distribution of Sales Channel
Amazon.in
Non-Amazon
0
5000
10000
15000
20000
25000


Distribution of Category
kurta
Set
Western Dress
Top
Blouse
Ethnic Dress
Bottom
Saree
0
2000
4000
6000
8000
10000


Distribution of Size
M
XL
L
XXL
S
3XL
XS
Free
6XL
5XL



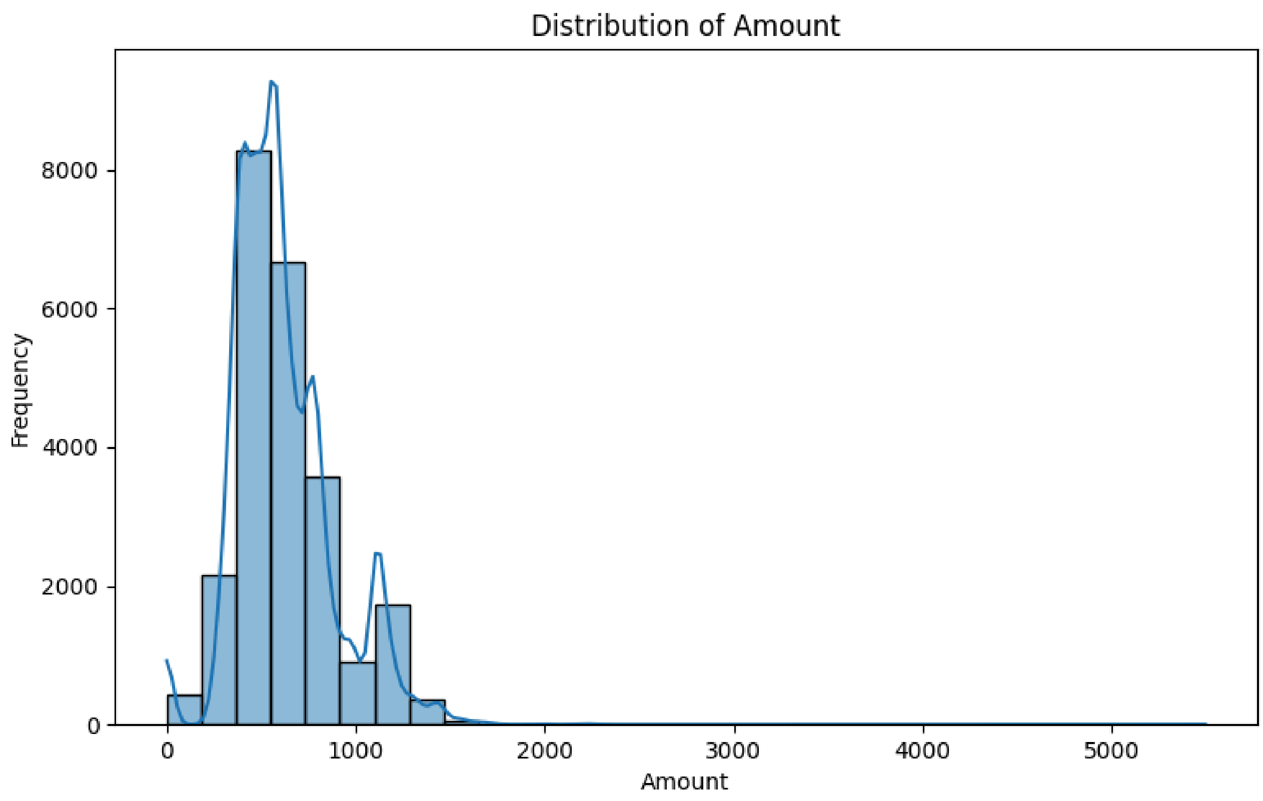


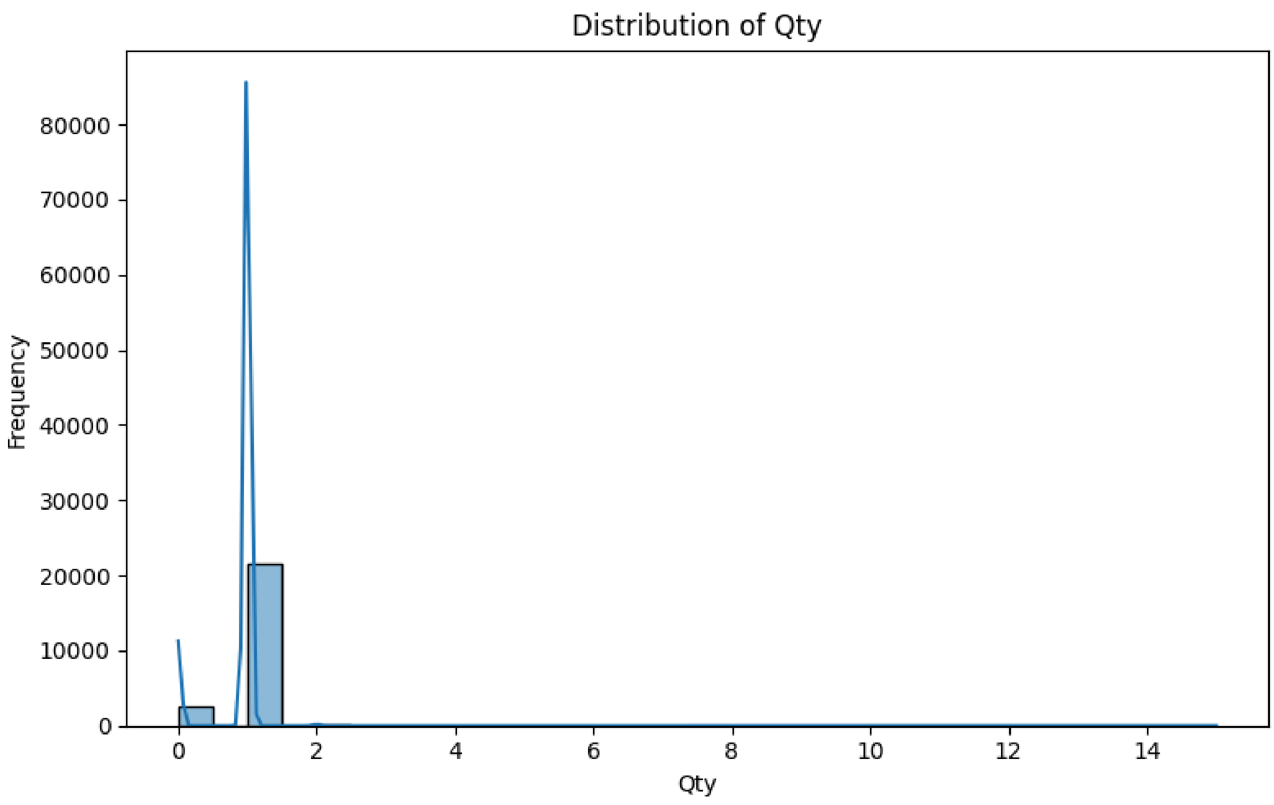
Numerical Data Analysis Outcomes

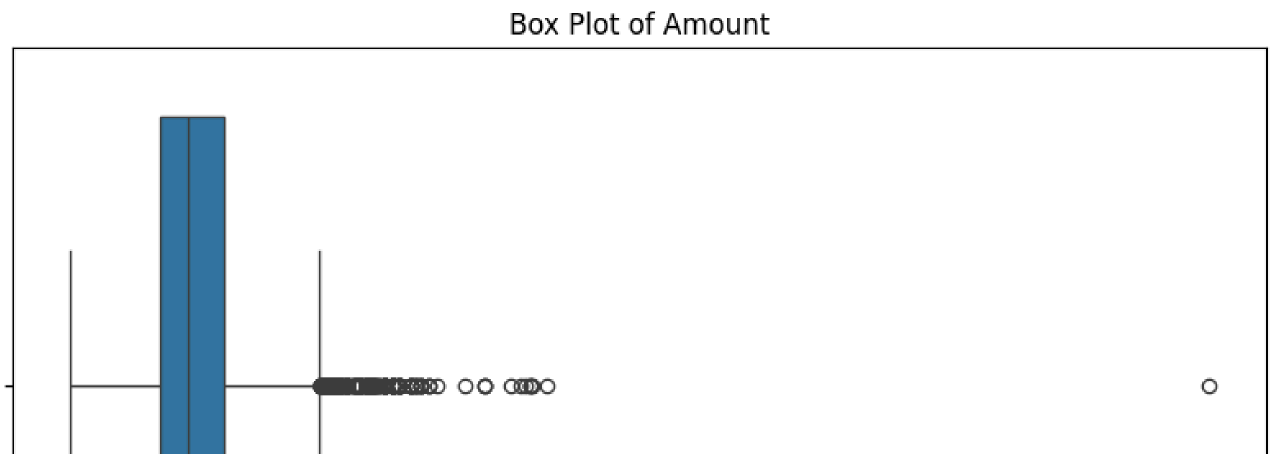
Distribution Insights

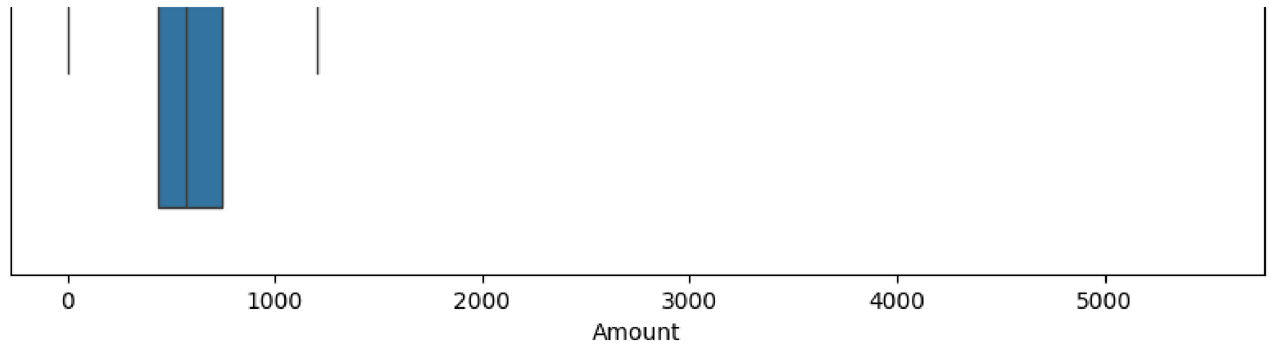
- Amount: The distribution of "Amount" is right-skewed, indicating that most transactions involve smaller amounts, with a few larger transactions. There are evident outliers in the amount data, as seen in the box plot, where some amounts are significantly higher than the general trend.
- Qty (Quantity): The majority of orders consist of small quantities, predominantly single items, as indicated by the peak at 1 in the histogram. The box plot confirms that larger quantities are rare but do occur, suggesting bulk purchases or possibly wholesale transactions.

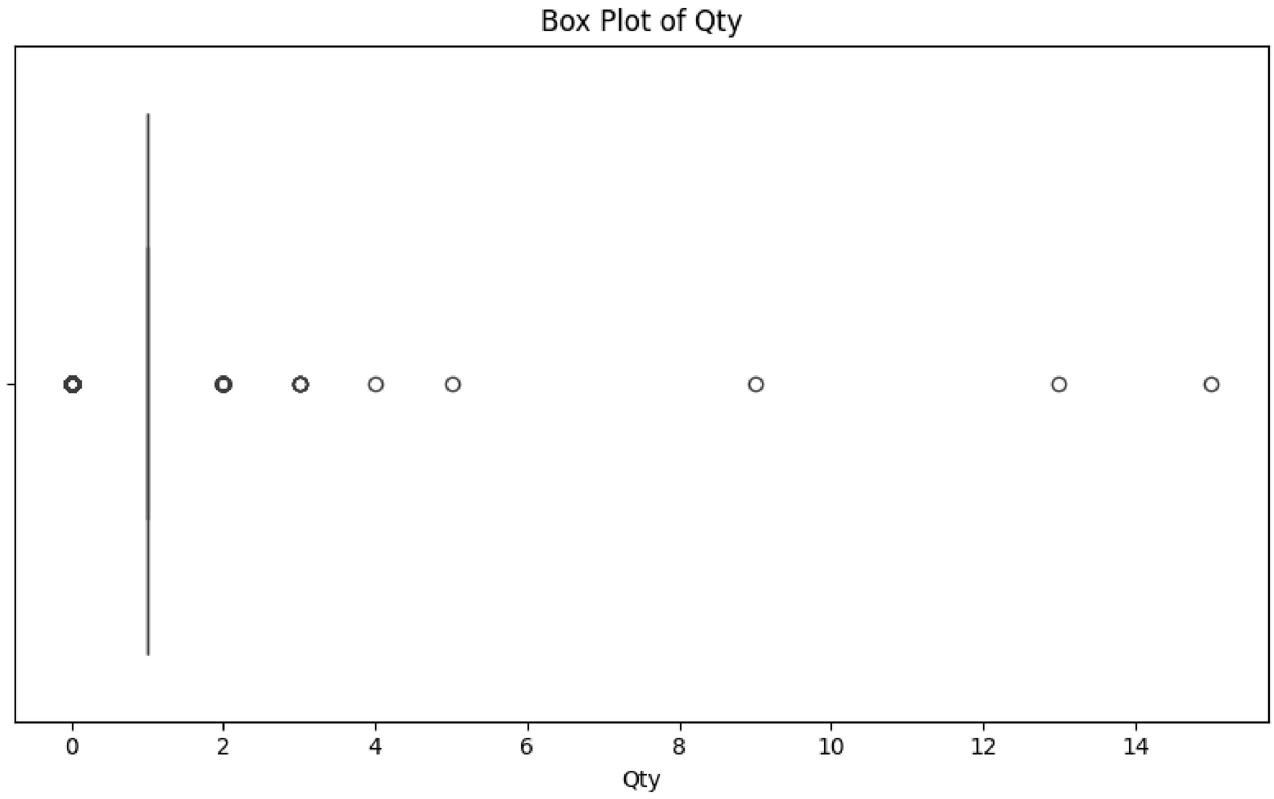
```python
# Plot numerical distributions and relationships
numerical_columns = ['Amount', 'Qty']

# Distribution plots for numerical variables
fig, axes = plt.subplots(nrows=len(numerical_columns), ncols=1, figsize=(8, 10))
for i, col in enumerate(numerical_columns):
    sns.histplot(data[col], bins=30, ax=axes[i], kde=True)
    axes[i].set_title(f'Distribution of {col}')
    axes[i].set_ylabel('Frequency')
    axes[i].set_xlabel(col)

plt.tight_layout()
plt.show()

# Boxplot to identify outliers
fig, axes = plt.subplots(nrows=len(numerical_columns), ncols=1, figsize=(8, 10))
for i, col in enumerate(numerical_columns):
    sns.boxplot(x=data[col], ax=axes[i])
    axes[i].set_title(f'Box Plot of {col}')

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


Distribution of Amount
Frequency
Amount
0
2000
4000
6000
8000
1000
3000
5000


Distribution of Qty
Frequency
Qty
0
10000
20000
30000
40000
50000
60000
70000
80000
2
4
6
8
10
12
14


Box Plot of Amount





ANOVA Test Outcomes

The ANOVA test reveals the following:

- F-statistic: 9715.17, which is significantly high, indicating that there is a strong effect of the category on the order amounts.
- P-value: Close to 0, suggesting that the differences in average amounts across categories are statistically significant.

- Interpretation: The very low p-value (PR(>F) < 0.001) confirms that there are significant differences in the mean order amounts among different categories. This implies that the type of product (category) has a strong influence on the transaction amount, which could be due to varying prices or consumer spending patterns in different product categories.

```
import statsmodels.api as sm
from statsmodels.formula.api import ols

# Drop NaN values in 'Amount' and 'Category' for ANOVA analysis
anova_data = data[['Amount', 'Category']].dropna()

# ANOVA model
anova_model = ols('Amount ~ C(Category)', data=anova_data).fit()

# ANOVA table
anova_table = sm.stats.anova_lm(anova_model, typ=2)
anova_table
```

	sum_sq	df	F	PR(>F)
C(Category)	7.072252e+08	7.0	2536.408847	0.0
Residual	9.613239e+08	24134.0	NaN	NaN

Next steps: Generate code with anova_table | View recommended plots | New interactive sheet

Analysis of Correlation Matrix

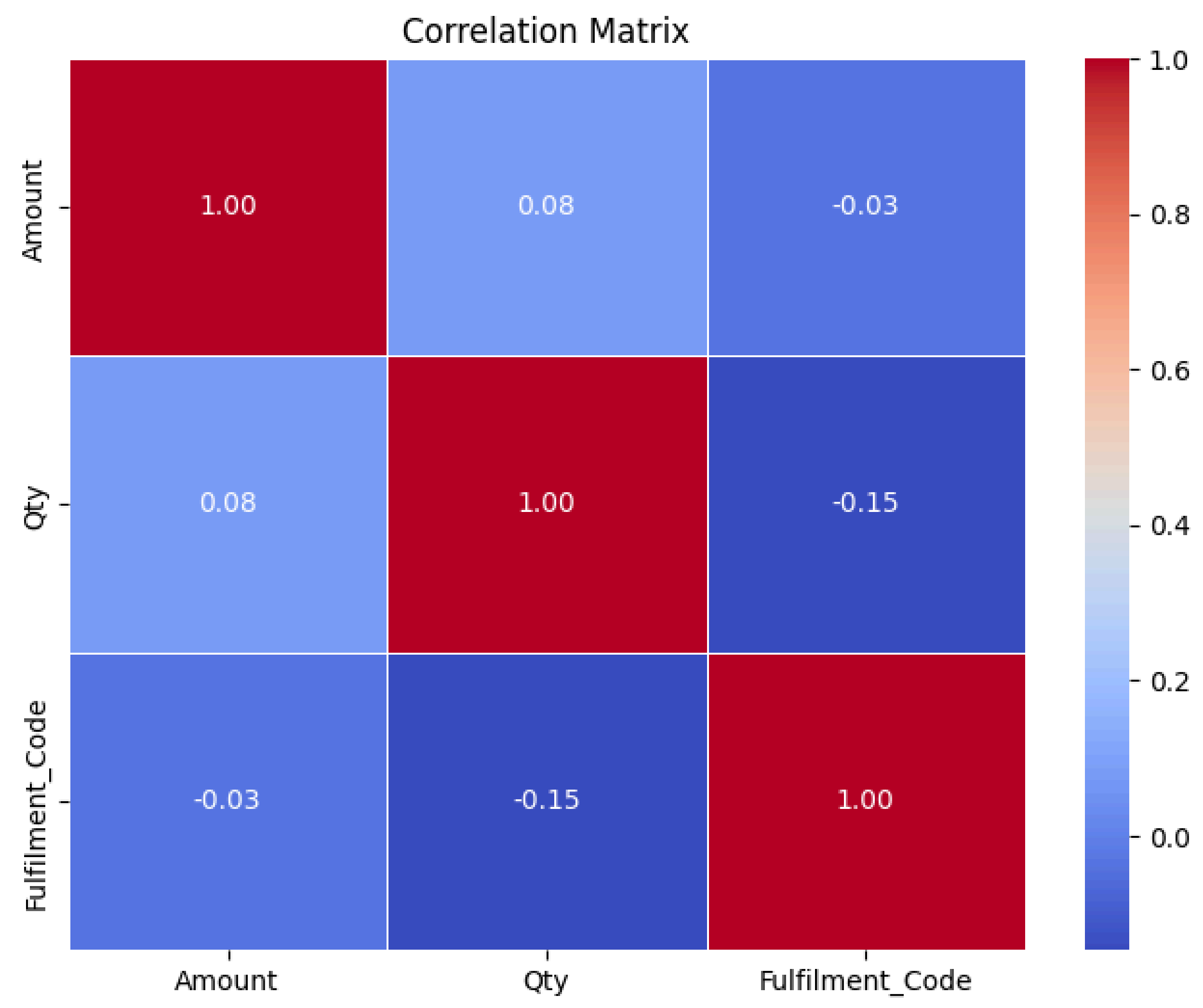
Amount vs. Qty:

- Correlation Coefficient: 0.08. This indicates a very weak positive relationship between the order amount and the quantity of items per order. It suggests that as the quantity of items in an order increases, there may be a slight increase in the total amount of the order, but the effect is minimal. Amount vs. Fulfilment_Code:

- Correlation Coefficient: -0.03. There is a negligible negative correlation between the order amount and the fulfilment method (coded as Fulfilment_Code). This implies that the method of fulfilment has almost no influence on the order amount, indicating uniform pricing across different fulfilment options.

- Qty vs. Fulfilment_Code: Correlation Coefficient: -0.15. This moderate negative correlation suggests that orders with higher quantities tend to be fulfilled differently than those with fewer quantities. It might indicate that bulk orders or larger quantities are handled differently, possibly due to logistical reasons or specific fulfilment strategies (like direct shipping from suppliers or specific warehouses).

```
# Convert 'Fulfilment' into a numeric code for correlation analysis
data['Fulfilment_Code'] = data['Fulfilment'].astype('category').cat.codes

# Select numerical and newly converted categorical data
correlation_data = data[['Amount', 'Qty', 'Fulfilment_Code']]

# Compute the correlation matrix
correlation_matrix = correlation_data.corr()

# Plot the correlation matrix
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=.5)
plt.title('Correlation Matrix')
plt.show()
```



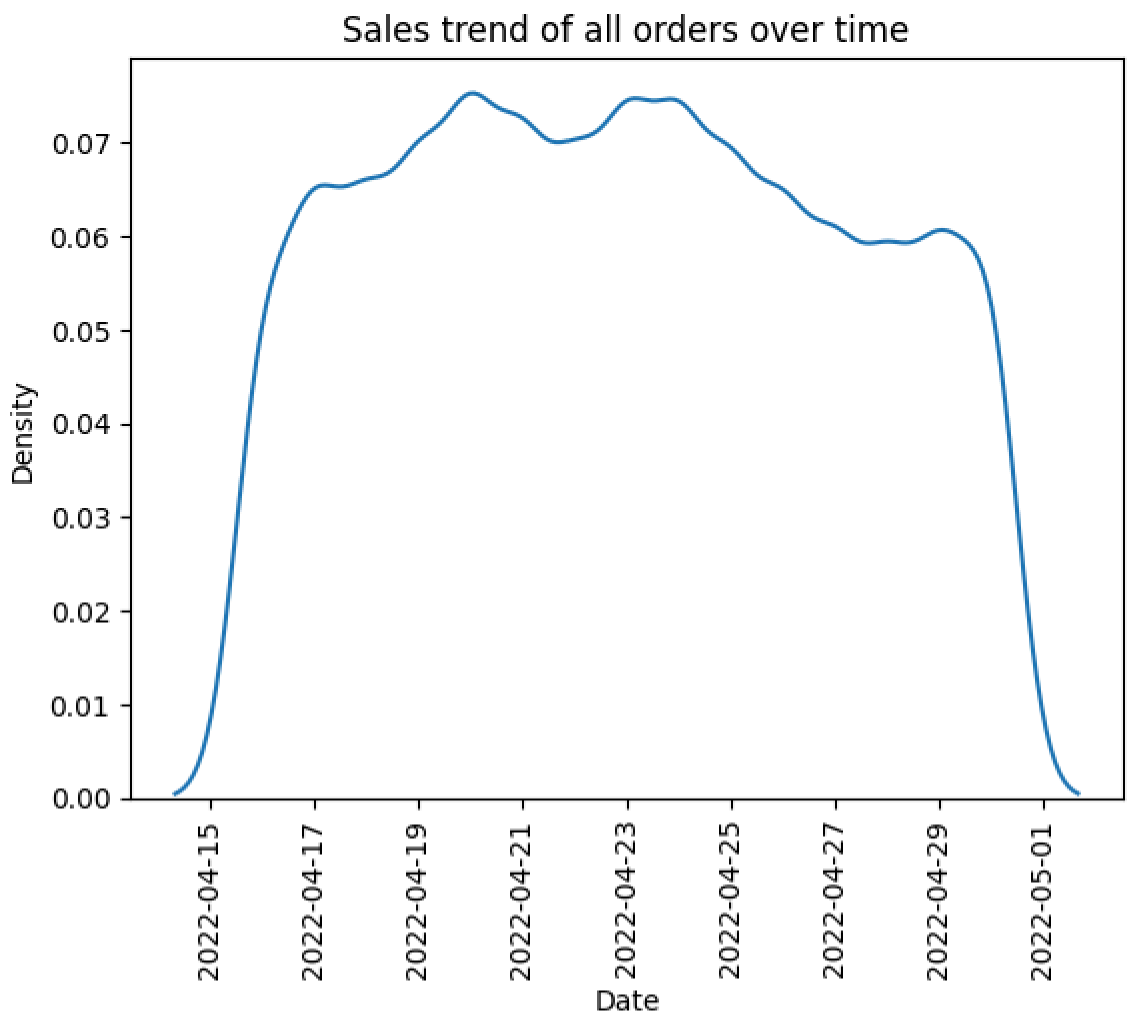
Analysis of the Sales Trend

1. **Rapid Increase in Sales**:
    - Initially, there's a sharp rise in the sales density from mid-April, reaching a peak towards the end of April. This suggests a significant increase in order activity, possibly driven by specific marketing campaigns, seasonal promotions, or other external factors such as holidays or special events.
2. **Stability at the Peak**:
    - The sales density stabilizes at the peak for a brief period, indicating sustained high activity levels. This plateau may reflect a period where sales efforts were maximized, and customer engagement was particularly high.
3. **Sharp Decline**:
    - Following the peak, there is a steep decline in sales density as we move into May. This drop could be due to the end of a promotion period, changes in consumer demand, or possibly inventory issues that could have affected order fulfillment.

Implications

- **Marketing and Promotions**: The peak in sales density could coincide with effective marketing strategies or promotions. Identifying these activities can help replicate successful tactics in future campaigns.
- **Inventory Management**: The sharp decline after the peak suggests that maintaining sufficient stock levels to meet consumer demand is crucial. It's also essential to ensure that operational efficiencies are managed to sustain high sales periods.
- **Consumer Behavior**: Understanding the factors influencing the buying patterns during this period could provide insights into consumer behavior, aiding in better targeting and customer engagement strategies in future.

```
sns.kdeplot(data=data,x="Date")
plt.xticks(rotation=90)
plt.title('Sales trend of all orders over time')
plt.show()
```



```
data.head()
```

	index	Order ID	Date	Status	Fulfilment	Sales Channel	ship-service-level	Style	SK
0	0	405-8078784-5731545	2022-04-30	Cancelled	Merchant	Amazon.in	Standard	SET389	SET38 KR-NP
1	1	171-9198151-1101146	2022-04-30	Shipped - Delivered to Buyer	Merchant	Amazon.in	Standard	JNE3781	JNE378 KR-XX
2	2	404-0687676-7273146	2022-04-30	Shipped	Amazon	Amazon.in	Expedited	JNE3371	JNE337 KR-X
3	3	403-9615377-8133951	2022-04-30	Cancelled	Merchant	Amazon.in	Standard	J0341	J034 DR
4	4	407-1069790-7240320	2022-04-30	Shipped	Amazon	Amazon.in	Expedited	JNE3671	JNE367 TU-XX

5 rows × 22 columns

```
cleaned_data=data.copy()
```

```
cleaned_data.to_csv('cleaned_data.csv',sep=',')
```

Extending Methods with Neural Networks

To further enhance the predictive capabilities of our e-commerce transaction analysis, incorporating a neural network model offers a powerful addition. Neural networks are capable of capturing complex nonlinear relationships that might be missed by more traditional machine learning approaches.

Neural Network Introduction

Neural Network Model: A type of deep learning model that consists of layers of interconnected nodes or neurons, which can learn to recognize patterns in data through backpropagation and a substantial amount of training data. They are particularly effective for large datasets with complex relationships and interactions.

Justification for Using a Neural Network

Advantages of Neural Networks:

- **Complex Pattern Recognition**: Neural networks are highly adept at identifying complicated patterns in data, making them suitable for the multifaceted nature of e-commerce transactions where interactions between variables can be non-linear and intricate.

- **Scalability**: They scale well with increasing amounts of data, which is typical in dynamic e-commerce environments where new data is continually generated.

Why It's Appropriate:

- Given the rich and diverse dataset available, a neural network can learn from the breadth of input features to better predict order statuses, potentially outperforming simpler models when configured and trained appropriately.
- Neural networks can integrate a mix of numeric, categorical, and even unstructured data, offering a comprehensive approach for the analysis.

Implementation Strategy for Neural Network

1. **Data Preprocessing**:
    - **Normalization**: Scale numerical inputs to ensure the neural network model converges efficiently.
    - **Encoding**: Convert categorical variables into one-hot encoded vectors to make them usable in the model.
2. **Model Architecture**:
    - **Input Layer**: Number of neurons equal to the number of features in the dataset.
    - **Hidden Layers**: Multiple hidden layers with a decreasing number of neurons, using ReLU activation functions to introduce non-linearity.
    - **Output Layer**: A single neuron with a sigmoid activation function for binary classification (cancelled vs not-cancelled) or multiple neurons with softmax for multi-class classification of order statuses.
3. **Training**:
    - **Backpropagation**: Use to adjust the weights in the network based on the error rate obtained in the previous epoch, ensuring learning from the training data.
    - **Regularization Techniques**: Implement dropout or L2 regularization to prevent overfitting.
    - **Optimizer**: Utilize adaptive learning rate optimizers like Adam for efficient training.
4. **Evaluation**:
    - **Cross-Validation**: Use to assess the generalizability of the model across different subsets of data.
    - **Performance Metrics**: Evaluate using accuracy, precision, recall, and F1-score.

Data Preprocessing: Purpose and Necessity

1. **Mapping Categorical Variables:**

   - **Why**: Machine learning models require numerical data. We convert categorical values (e.g., `Courier Status`, `Fulfilment`) to numerical formats so the model can process them effectively.

2. **Handling Missing Values:**
   - **Why**: Missing data can disrupt model training. We fill missing `Courier Status` values with a placeholder (`-999`) to maintain data consistency.

3. **Data Splitting:**
   - **Why**: Splitting the data into training and testing sets allows us to evaluate model performance on unseen data, ensuring that the model can generalize well.

Model Training and Predictions

1. **Random Forest Model:**
   - **Purpose**: The Random Forest model is trained to predict the target variable using an ensemble of decision trees. It is evaluated on the test set to assess its predictive performance.

2. **Logistic Regression Model:**
   - **Purpose**: Logistic Regression is used to model the relationship between the features and the target variable, particularly effective for binary or categorical outcomes. The model's accuracy is then tested on the unseen data.

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split

def data_preparation(file_path, target_column='Courier Status', test_size=0.3, random_se
    dataset = pd.read_csv(file_path)
    b2b_mapping = {False: 0, True: 1}
    status_mapping = {'Shipped': 1, 'Unshipped': 0, 'Cancelled': -1}
    fulfilment_mapping = {'Amazon': 1, 'Merchant': 0}
    service_level_mapping = {'Expedited': 1, 'Standard': 0}
    dataset[target_column] = dataset[target_column].map(status_mapping)
    dataset['B2B'] = dataset['B2B'].map(b2b_mapping)
    dataset['Fulfilment'] = dataset['Fulfilment'].map(fulfilment_mapping)
    dataset['ship-service-level'] = dataset['ship-service-level'].map(service_level_mapp
    dataset[target_column] = dataset[target_column].fillna(-999).astype(int)
    feature_columns = ['Fulfilment', 'ship-service-level', 'Qty', 'Amount', 'ship-postal
    X_features = dataset[feature_columns]
    y_target = dataset[target_column]
    # Split the dataset into train and test sets
    X_train_set, X_test_set, y_train_set, y_test_set = train_test_split(X_features, y_ta
    return X_train_set, X_test_set, y_train_set, y_test_set
```

```
# Prepare the data
X_train, X_test, y_train, y_test = data_preparation('/content/cleaned_data.csv')


# Train the Random Forest model
model = RandomForestClassifier(n_estimators=100, max_depth=5, random_state=1)
model.fit(X_train, y_train)
# Make predictions
predictions = model.predict(X_test)




# Calculate and print accuracy
accuracy = accuracy_score(y_test, predictions)
print(f"RF Model Accuracy: {accuracy:.2f}")
print('_' * 30)
# Generate and print classification report
report = classification_report(y_test, predictions)
print(report)
```

```
RF Model Accuracy: 0.96
______________________________
              precision    recall  f1-score   support

        -999       1.00      1.00      1.00       482
          -1       1.00      1.00      1.00       339
           0       1.00      0.03      0.06       294
           1       0.96      1.00      0.98      6128

    accuracy                           0.96      7243
   macro avg       0.99      0.76      0.76      7243
weighted avg       0.96      0.96      0.94      7243
```

```
from sklearn.metrics import mean_squared_error , mean_absolute_error

model = LogisticRegression(penalty='l2', solver='lbfgs', C=0.1)

model.fit(X_train, y_train)
#testing the model
y_pred = model.predict(X_test)


# Perform K-Fold cross-validation
cv = KFold(n_splits=5, shuffle=True, random_state=45)

scores = cross_val_score(model, X_train, y_train, cv=cv, scoring='accuracy')
print("Cross-Validation Accuracy Scores:")
print(scores)
print('_'*30)
print(f"Mean Cross-Validation Accuracy: {scores.mean():.2f}")
```

```
Cross-Validation Accuracy Scores:
[0.85946746 0.85443787 0.86715976 0.84763314 0.86001776]
```

```
Mean Cross-Validation Accuracy: 0.86
```

Data Preprocessing for Neural network: Purpose and Necessity

1. **Mapping Categorical Variables:**
    - **Why**: Neural networks require numerical input. Categorical variables (like `Courier Status`, `Fulfilment`) are mapped to numerical values to enable the model to process them.
2. **Handling Missing or Invalid Values:**
    - **Why**: To ensure the data is clean and consistent, any missing or invalid values in `Courier Status` are removed. This step helps maintain the integrity of the dataset and ensures the model only trains on valid data.
3. **Transforming the Target Variable:**
    - **Why**: The `Courier Status` target variable is transformed to a numerical range (0 to n_classes - 1) suitable for classification tasks. This step is crucial for preparing the target variable for the neural network model.
4. **Splitting the Data:**
    - **Why**: The dataset is split into training and testing sets to evaluate the model's performance on unseen data. This split helps in assessing the model's generalization ability.
5. **Standardizing Features:**
    - **Why**: Standardization scales the feature values, ensuring they have a mean of 0 and a standard deviation of 1. This is important for neural networks, as it helps speed up the training process and can lead to better performance.

Neural Network Model Training and Evaluation

1. **Building the Neural Network:**
    - **Purpose**: A neural network model is constructed with layers that consist of neurons. The first two layers are hidden layers with `ReLU` activation functions, which introduce non-linearity and help the network learn complex patterns. The output layer uses `softmax` activation, suitable for multi-class classification.
2. **Compiling the Model:**

   - **Purpose**: The model is compiled using the `sparse_categorical_crossentropy` loss function, which is ideal for multi-class classification with integer labels. The optimizer `adam` is used for its efficiency in training, and accuracy is chosen as the evaluation metric.
3. **Training the Model:**
   - **Purpose**: The model is trained on the training data (`X_train`, `y_train`) for a specified number of epochs. During training, the model learns to map input features to the target classes. A validation split is used to monitor the model's performance on unseen data during training.
4. **Evaluating the Model:**
   - **Purpose**: After training, the model's performance is evaluated on the test set (`X_test`, `y_test`). The test accuracy indicates how well the model generalizes to new data.
5. **Generating Predictions:**
   - **Purpose**: Predictions are generated for the test set, which can be further analyzed to assess the model's performance or for making decisions based on the predicted outcomes.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

def prepare_nn_data(file_path):
    data = pd.read_csv(file_path) # Load the cleaned dataset
    B2B_map = {False: 0, True: 1} # Define the mapping dictionaries
    category_map = {'Shipped': 1, 'Unshipped': 0, 'Cancelled': -1}
    Fulfilment_map = {'Amazon': 1, 'Merchant': 0}
    ship_service_level_map = {'Expedited': 1, 'Standard': 0}
    # Apply the mappings to the corresponding columns
    data['Courier Status'] = data['Courier Status'].map(category_map)
    data['B2B'] = data['B2B'].map(B2B_map)
    data['Fulfilment'] = data['Fulfilment'].map(Fulfilment_map)
    data['ship-service-level'] = data['ship-service-level'].map(ship_service_level_map)
    data = data[data['Courier Status'].isin([0, 1, -1])]  # Only keep valid categories
    label_map = {-1: 0, 0: 1, 1: 2}
    data['Courier Status'] = data['Courier Status'].map(label_map)
    # Define the feature columns and the target
    feature_cols = ['Fulfilment', 'ship-service-level', 'Qty', 'Amount', 'ship-postal-co
    X = data[feature_cols]
    y = data['Courier Status']


    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_stat

    # Standardize the features
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    return X_train, X_test, y_train, y_test



from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

def build_and_train_nn(X_train, y_train, X_test, y_test):
    model = Sequential()
    model.add(Dense(32, input_dim=X_train.shape[1], activation='relu'))  # First hidden 
    model.add(Dense(16, activation='relu'))  # Second hidden layer with 16 neurons
    model.add(Dense(len(y_train.unique()), activation='softmax'))  # Output layer with s
    # Compile the model
    model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['ac
    model.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=0.2)
    # Evaluate the model
    test_loss, test_accuracy = model.evaluate(X_test, y_test)
    print(f'NN Model Test Accuracy: {test_accuracy:.2f}')

    y_pred = model.predict(X_test)

    return model, y_pred
```

```
# File path to your cleaned data
file_path = '/content/cleaned_data.csv'
X_train, X_test, y_train, y_test = prepare_nn_data(file_path)
model, y_pred = build_and_train_nn(X_train, y_train, X_test, y_test)
```

```
Epoch 1/10
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarni
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
396/396 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9155 - loss: 0.4351 - val_acc
Epoch 2/10
396/396 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9574 - loss: 0.1632 - val_acc
Epoch 3/10
396/396 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9640 - loss: 0.1389 - val_acc
Epoch 4/10
396/396 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9597 - loss: 0.1505 - val_acc
Epoch 5/10
396/396 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.9603 - loss: 0.1442 - val_acc
Epoch 6/10
396/396 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9573 - loss: 0.1553 - val_acc
Epoch 7/10
396/396 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9611 - loss: 0.1459 - val_acc
Epoch 8/10
396/396 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9598 - loss: 0.1500 - val_acc
Epoch 9/10
396/396 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9613 - loss: 0.1454 - val_acc
Epoch 10/10
396/396 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9638 - loss: 0.1387 - val_acc
212/212 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9602 - loss: 0.1484
NN Model Test Accuracy: 0.96
212/212 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
```

Model Evaluation

Evaluation Metrics

To assess the performance of our models, we utilized several key metrics:

- **Accuracy**: This metric measures the overall correctness of the model, representing the ratio of correctly predicted instances to the total instances in the dataset.
- **Precision**: Precision measures the accuracy of the positive predictions. It is defined as the ratio of true positive predictions to the sum of true positive and false positive predictions.
- **Recall**: Recall, also known as sensitivity or true positive rate, measures the model's ability to correctly identify positive instances.
- **F1-Score**: The F1-score is the harmonic mean of precision and recall, providing a balanced measure when there is an uneven class distribution.

Model Performance

1. Random Forest

The Random Forest model achieved an accuracy of 96%. Below are the detailed metrics:

- **Precision**: The model exhibited high precision across most classes, with perfect precision for the majority of classes, except for class `0`, where the precision was lower due to imbalanced data.
- **Recall**: The recall for the class `1` (representing 'Shipped') was particularly high, indicating that the model was very effective in identifying the shipped orders. However, recall for class `0` was notably low.
- **F1-Score**: The F1-score for most classes was high, reflecting the model's strong ability to balance precision and recall effectively.

The model struggled slightly with class `0`, possibly due to the data imbalance, leading to a lower F1-score for that category.

2. Logistic Regression

Logistic Regression was evaluated using 5-fold cross-validation, resulting in a mean accuracy of 86%. The cross-validation scores demonstrated the model's consistency, with accuracy values ranging from approximately 85% to 87%.

This performance indicates that Logistic Regression, while effective, may not capture the complex patterns in the data as well as the Random Forest model. It's a more straightforward model, which can be advantageous for interpretation but may lack the predictive power of more complex models.