BCIS5110-Programming Languages for Business Analytics

Project Final Report

Reported By:

Venkateswara Reddy Lingareddygari

Professor: Ling Ge

Contents

Executive Summary:

The executive summary offers a concise overview of the analysis conducted on JD.com's extensive e-commerce data set. Committed to promoting data-driven research in E-tailing, JD.com, as China's foremost retailer, has provided a dataset that captures the entire customer experience cycle. This dataset, authored by Max Shen, Christopher S. Tang, Di Wu, Rong Yuan, and Wei Zhou, spans the month of March 2018, presenting insights from over 2.5 million customers and 30,000 SKUs.

JD.com, boasting a remarkable net revenue of US$67.2 billion in 2018 and an expansive customer base exceeding 320 million annual active customers, stands as a leader in China's retail landscape. The company is recognized for its steadfast dedication to delivering high-quality, genuine products with exceptional speed.

JD.com's business model integrates a first-party approach, retaining control over the entire supply chain, and a curated marketplace that restricts the number of sellers. This meticulous strategy ensures stringent quality oversight. The nationwide fulfillment network covers 99% of China's population, facilitating standard same- and next-day delivery for approximately 90% of orders.

The dataset comprises information on 2.5 million customers engaging with 30,000 SKUs within a specific product category throughout March 2018, providing a detailed snapshot of the e-commerce terrain.

Project Motivation/Background:

JD.com, positioned as the foremost retailer in China, occupies a significant role in the country's retail sector, disclosing a noteworthy net revenue of US$67.2 billion in 2018 and proudly presenting an extensive customer base exceeding 320 million active customers annually. The official depiction provided by JD.com underscores the company's steadfast dedication to delivering high-quality, genuine products, coupled with a reputation for prompt and dependable delivery services.

Renowned for establishing the benchmark in online shopping, JD.com's business model is distinguished by a dual approach. The first-party model ensures complete control over the entire supply chain, supplemented by a curated marketplace deliberately restricting the number of sellers. This strategic fusion is crafted to maintain rigorous quality oversight, reinforcing JD.com's commitment to product authenticity and excellence across a varied spectrum of offerings, spanning from fresh produce and clothing to electronics and beauty products.

The datasets graciously provided by JD.com encapsulate a thorough "full customer experience cycle." Initiated from the point a customer explores products on the platform to the order placement and culminating in the receipt of products at the designated location, these datasets afford a detailed perspective into customer interactions. Specifically, the data encompasses details on 2.5 million customers engaging with 30,000 SKUs within a specific product category during the month of March in 2018.

The following questions made us motivated in working with this project.

1. What is the peak time for orders during a day and also find the customer order trend over a day's time.
2. Which price range has the most orders?
3. What is the education level of the majority?
4. Which age level has the most users?

In the below, we are predicting the average coupon discount per order.

```
#1.What is the average coupon discount per order? the given product category?
avg_discount_by_category = df_order.groupby('type')['coupon_discount_per_unit'].mean()
print(avg_discount_by_category)
```

```
type
1    2.321762
2    3.532353
Name: coupon_discount_per_unit, dtype: float64
```

Data Description:

The provided JD.com transactional data offers a detailed perspective on the entire customer journey, spanning from product exploration to delivery. This dataset, concentrated on a specific product category in March 2018, comprises seven distinct tables: skus, users, clicks, orders, delivery, inventory, and network.

Skus Table:

Profiles 31,868 SKUs, detailing critical attributes like sku_ID, type (1P or 3P), brand_ID, and specific attributes.
Underlines the significance of first-party (1P) SKUs under JD.com's control, emphasizing quality assurance and enhanced customer experience.
Provides comprehensive insights into SKU characteristics, activation, and deactivation dates.

Users Table:

Characterizes 457,298 users, encompassing details such as user_ID, user_level, first_order_month, and demographic approximations.
Distinguishes users based on factors like PLUS membership, education, age, and purchasing power.
Integrates estimated user demographics and pertinent shipping address details.

Orders Table:

Encompasses 486,928 customer orders, including order-specific specifics such as order_ID, user_ID, sku_ID, and intricate pricing details.
Discriminates between 1P and 3P orders, offering insights into discounts, quantity dynamics, and assured delivery times.

Delivery Table:

Establishes a clear link between orders and delivery packages, tracking shipment times and categorizing order types (1P or 3P).
Logs details like package_ID, order_ID, ship_out_time, arr_station_time, and arr_time for 293,229 packages facilitated by JD Logistics.

Inventory Table:

Field	Data type	Description	Sample value
dc_ID	int	Distribution center ID	9
sku_ID	string	SKU unique identification code	fcc883f713
date	string	Date (format: yyyy-mm-dd)	2018-03-01

Offers comprehensive insights into SKU availability across various warehouses on specific dates.

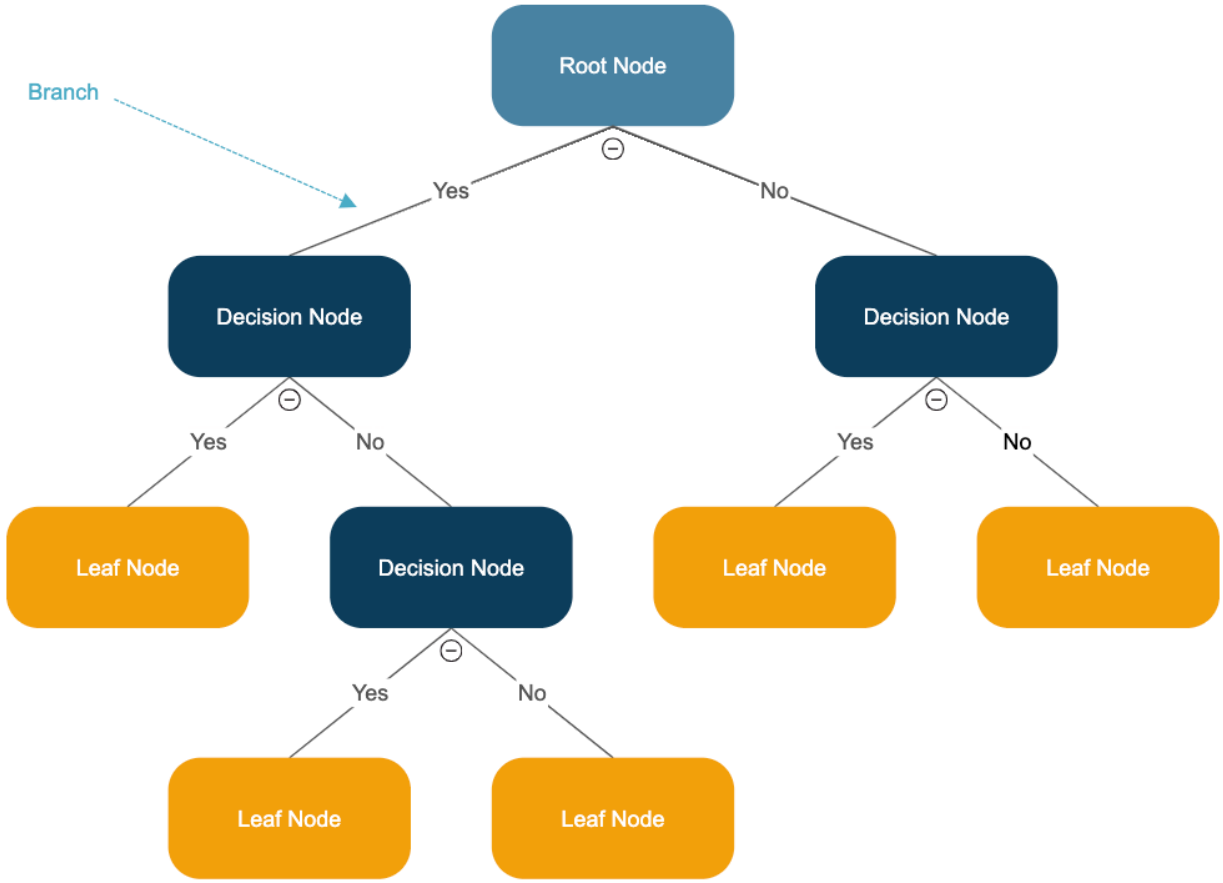
Network Table:

Field	Data type	Description	Sample value
region_ID	int	Region ID	2
dc_ID	int	District ID (same as warehouse ID)	6

Furnishes data regarding the allocation of warehouses to regions, designating central warehouses for backup fulfillment.

Model and Analysis:

A decision tree is a type of tree structure that resembles a flowchart in which each leaf node represents the outcome, and each inner node denotes a feature (or attribute). The node at the top of a decision tree is known as the root node. Data division depending on the value of an attribute becomes possible. The name signifies that recursive partitioning divides the tree in several ways. This framework, which resembles a flowchart, makes decision-making simpler. To mimic human thought, it employs visualization, much like a flowchart diagram. As a result, decision trees are easy to understand and interpret.



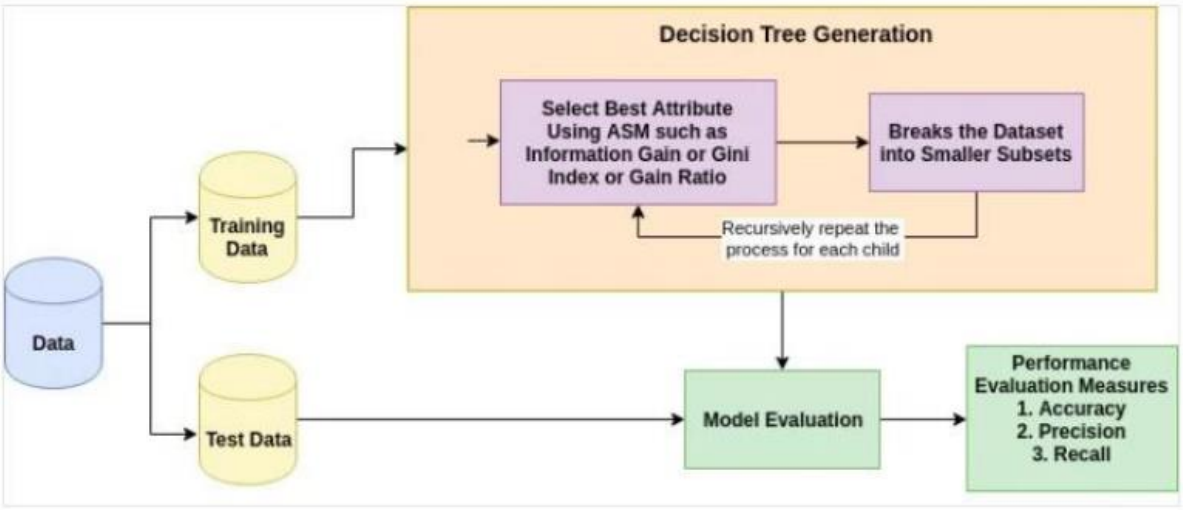
Predictions are made on the test data, and the model is evaluated using the Root Mean Squared Error (RMSE). The RMSE is calculated and displayed, indicating the model's performance. Finally, the statistics of the target variable, 'delivery_time,' are examined to provide context to the model's predictions. The entire process provides a comprehensive

approach to aggregating and analyzing order data, culminating in the assessment of a Decision Tree regression model's predictive accuracy.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
from sklearn.tree import DecisionTreeRegressor
regressor = DecisionTreeRegressor(random_state=42)
regressor.fit(X_train, y_train)
predictions = regressor.predict(X_test)
predictions
from sklearn.metrics import mean_squared_error
import numpy as np
mse_test = mean_squared_error(y_test, predictions)
rmse_test = np.sqrt(mse_test)
print(f"Root Mean Squared Error on Testing Data: {rmse_test:.2f}")
```

```
Root Mean Squared Error on Testing Data: 27.00
```

Decision Tree Algorithm:



Using Decision Tree Classifier on the JD Dataset dataset using Scikit-learn Import the required libraries

Data Preparation:

A critical step in the workflow for data analysis and machine learning is data preparation, sometimes referred to as data preprocessing or data cleaning. To prepare raw data for analysis or machine learning model training, it must be cleaned and transformed. Making sure the data is precise, comprehensive, and well-structured is the aim of data preparation since it enables the derivation of valuable insights and the efficient training of models.

Here are some common tasks involved in data preparation:

Data cleaning: The process of identifying and correcting or handling errors, inconsistencies, and inaccuracies in datasets. The goal of data cleaning is to improve the quality of the data, making it suitable for analysis, reporting, and machine learning applications.

Removing unwanted data: It is a key step in the data cleaning process. Unwanted data can include irrelevant columns, duplicate records, or any information that does not contribute to the analysis or modeling objectives.

Data Merging: The process of combining information from several datasets into one single dataset is called data merging, sometimes referred to as data integration or joining, in data preparation. Usually, this is carried out using common columns or keys that are present in both datasets. The objective is to produce a single, comprehensive dataset with all the pertinent data required for reporting, analysis, or modeling. some key points and considerations for data merging Inner merge, right merge, and left merge.

Data cleaning:

In below analysis we need to clean the merged order and delivery table. We display the unique values of the **type_x** and **type_y** variables to understand their current values. This helps us identify the unique categories present in each variable. After cleaning, we display the updated unique values of **type_x** and **type_y** to confirm that the two variables are now consistent. They should now have the same set of values based on the assumed data description.

```
# Assuming your DataFrame is named 'merged_table'

# Check if values in 'type_x' are the same as values in 'type_y'
are_types_equal = (merged_data_inner['type_x'] == merged_data_inner['type_y']).all()

# Print the result
if are_types_equal:
    print("The values of 'type_x' and 'type_y' are the same.")
    print(are_types_equal)
else:
    print("The values of 'type_x' and 'type_y' are not the same.")
    print(are_types_equal)
```

```
The values of 'type_x' and 'type_y' are the same.
True
```

Since, they are same we are dropping column “type_y”.

```
inner_merged.drop(columns=['type_y'], inplace=True)
print("Unique values of 'type_x' after replacement:")
print(inner_merged['type_x'].unique())
```

```
Unique values of 'type_x' after replacement:
[1 0]
```

Data removing:

In below analysis we need to remove the orders that has a single item, and the item is a gift item. To identify single-item orders, the code creates a Boolean mask (single_item_orders) in the inner_merged DataFrame for non-duplicated order IDs. The DataFrame is then filtered to keep only those orders, and the outcome is stored in filtered_orders. For orders containing a single item, the count of observations and unique order IDs are shown in the following lines. The number of observations, or orders with a single item, in the filtered_orders DataFrame is printed by the code. Subsequently, it generates a fresh Data Frame called filtered_data by eliminating rows in which the 'gift_item' column contains True. Once single gift item orders are eliminated, it prints the updated number of observations in filtered_data.

```
single_item_orders = ~inner_merged['order_ID'].duplicated(keep=False)
filtered_orders = inner_merged[single_item_orders]
print("Number of observations for orders with a single item:", len(filtered_orders))
print("Number of unique orders with a single item:", len(filtered_orders['order_ID'].unique()))
```

```
Number of observations for orders with a single item: 265209
Number of unique orders with a single item: 265209
```

Then we filter the data to remove those orders of a single gift item. Save the changes.

```
print("Number of observations for orders with a single item:", len(filtered_orders))
filtered_data = filtered_orders[~(filtered_orders['gift_item'] == True)]
print("\nNumber of observations after removing single gift item orders:", len(filtered_data))
final_data = filtered_data
```

```
Number of observations for orders with a single item: 265209

Number of observations after removing single gift item orders: 252886
```

The explanation for this is that such orders could have originated from other product categories and only used products from the present category as gifts. We have no knowledge of those orders.

Data merging:

In Below analysis we perform inner merge for order and delivery tables.

```
inner_merged = pd.merge(order_ab, delivery_ab, on='order_ID')
print("Number of observations in inner merge:", len(inner_merged))
```

```
Number of observations in inner merge: 326862
```

Inner merge: In an inner merge, only the records that have matching values in the key column in both data frames are retained in the merged dataset.

Number of observations in inner merge: 326862

Number of observations in Order data: 549989

Number of observations in delivery data: 293229

After performing an inner merge, you end up with 326,862 records. This means that there are 326,862 instances where there are matching 'order_id' values in both the 'Order' and 'Delivery' data.

Right merge: Here we perform right merge for order and delivery tables.

```
inner_merged = pd.merge(order_ab, delivery_ab, on='order_ID')
print("Number of observations in inner merge:", len(inner_merged))
```

```
Number of observations in inner merge: 326862
```

Right Merge: In a right merge, all records from the right data frame are retained in the merged dataset, and only matching records from the left data frame are included.

"Number of observations in right merge: 326862"

Left Merge: Here we perform left merge for order and delivery tables.

```
left_merged = pd.merge(order_ab, delivery_ab, on='order_ID', how='left')
print("Number of observations in left merge:", len(left_merged))
```

```
Number of observations in left merge: 550017
```

Left Merge: All records from the left data frame are retained in the merged dataset, and only matching records from the right data frame are included.

Number of observations in left merge: 550017

Number of observations in Order data: 549989

Following a left merge, we have 550,017 records in the end. This indicates that in 550,017 cases, the order_id variables in the Order and Delivery data match.To guarantee that every entry from the 'Order' data is included in the combined dataset, a left merge is used.

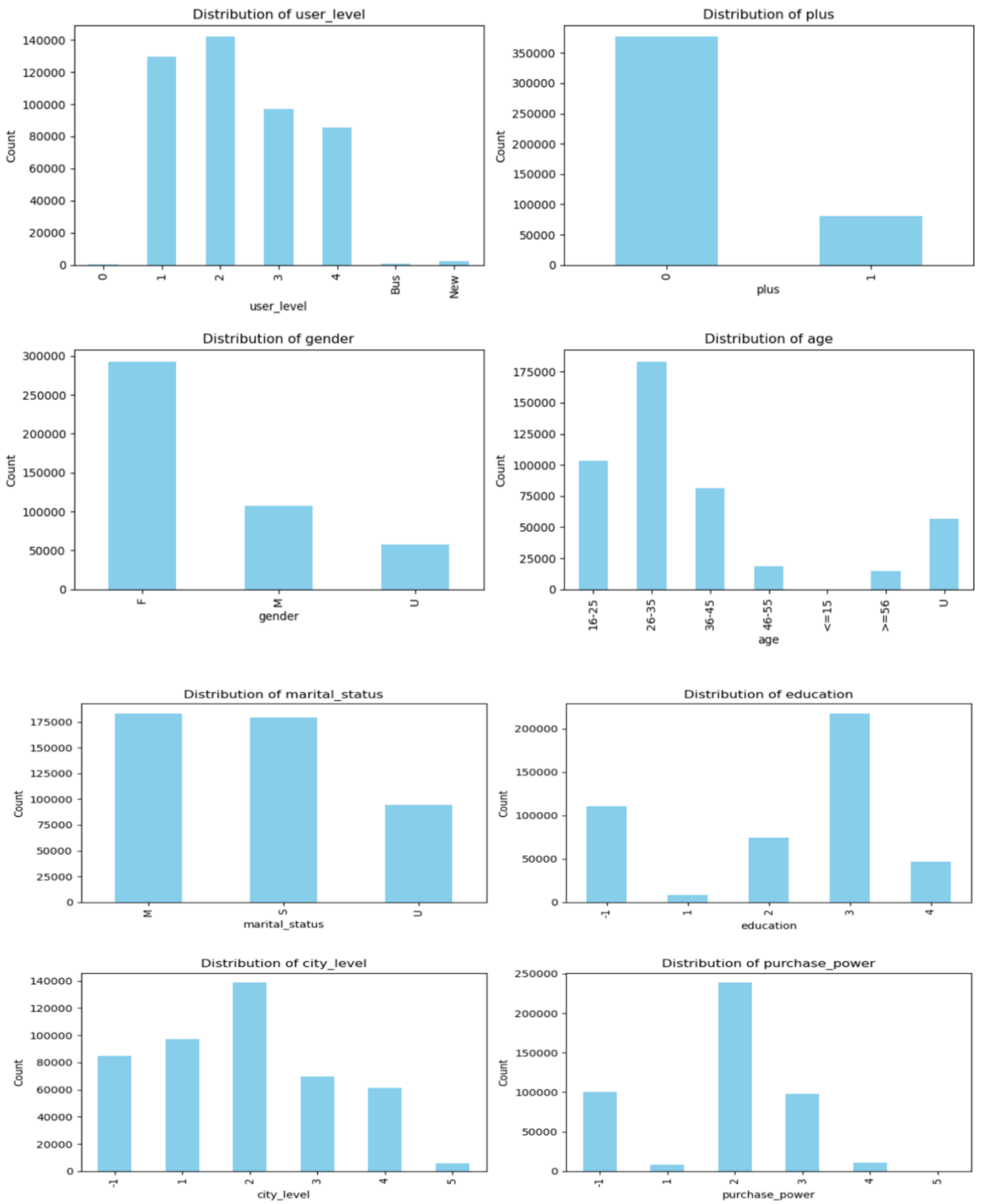
Exploratory Data Analysis:

Exploratory Data Analysis (EDA) is a crucial phase in the data analysis process that involves visually and statistically summarizing, interpreting, and understanding the main characteristics of a dataset. EDA helps analysts and data scientists gain insights into the underlying patterns, relationships, and trends in the data. To explain and comprehend the link between factors and how these can affect business, it makes use of a variety of statistical tools and data manipulation techniques.

Below are some of the exploratory analysis:

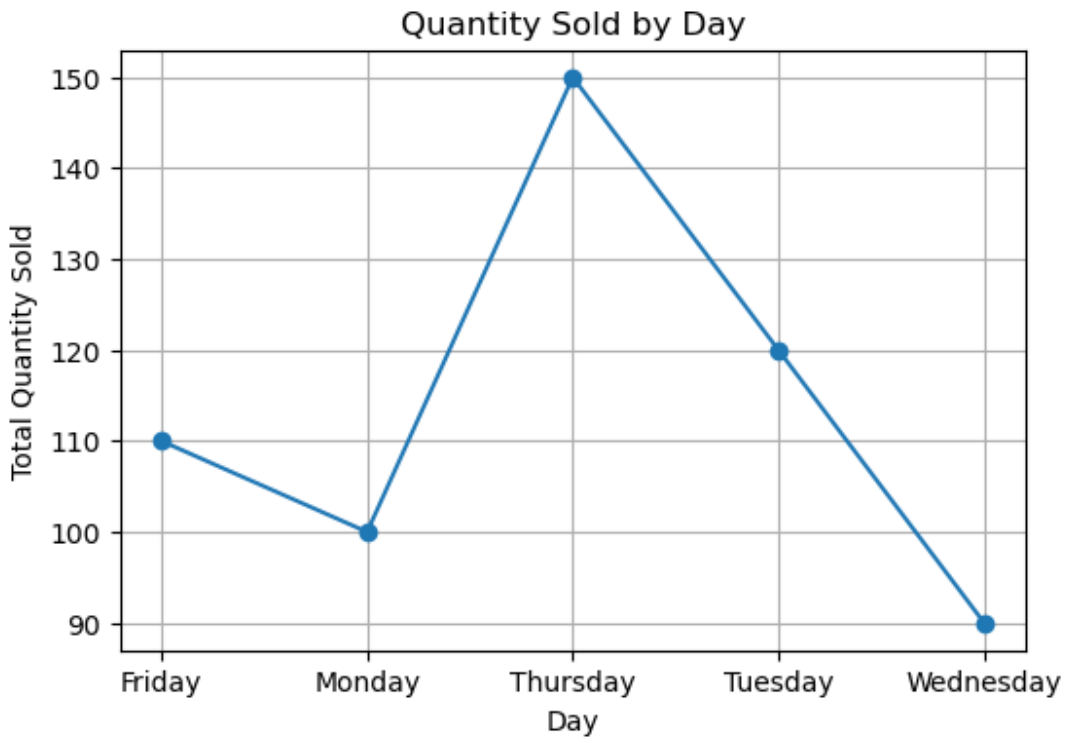
Below is the analysis of distribution of "user_level', 'plus', 'gender', 'age', 'marital_status', 'education', 'city_level', and 'purchase_power'.

```
import matplotlib.pyplot as plt
categorical_variables = ['user_level', 'plus', 'gender', 'age', 'marital_status', 'education', 'city_lev
fig, axes = plt.subplots(nrows=4, ncols=2, figsize=(12, 16))
axes = axes.flatten()
for i, variable in enumerate(categorical_variables):
    user_ab[variable].value_counts().sort_index().plot(kind='bar', ax=axes[i], color='skyblue')
    axes[i].set_title(f'Distribution of {variable}')
    axes[i].set_xlabel(variable)
    axes[i].set_ylabel('Count')
plt.tight_layout()
plt.show()
```


Distribution of user_level
Count
user_level
0
1
2
3
4
Bus
New
Distribution of plus
Count
plus
0
1
Distribution of gender
Count
gender
F
M
U
Distribution of age
Count
age
16-25
26-35
36-45
46-55
<=15
>=56
U
Distribution of marital_status
Count
marital_status
M
S
U
Distribution of education
Count
education
-1
1
2
3
4
Distribution of city_level
Count
city_level
-1
1
2
3
4
5
Distribution of purchase_power
Count
purchase_power
-1
1
2
3
4
5

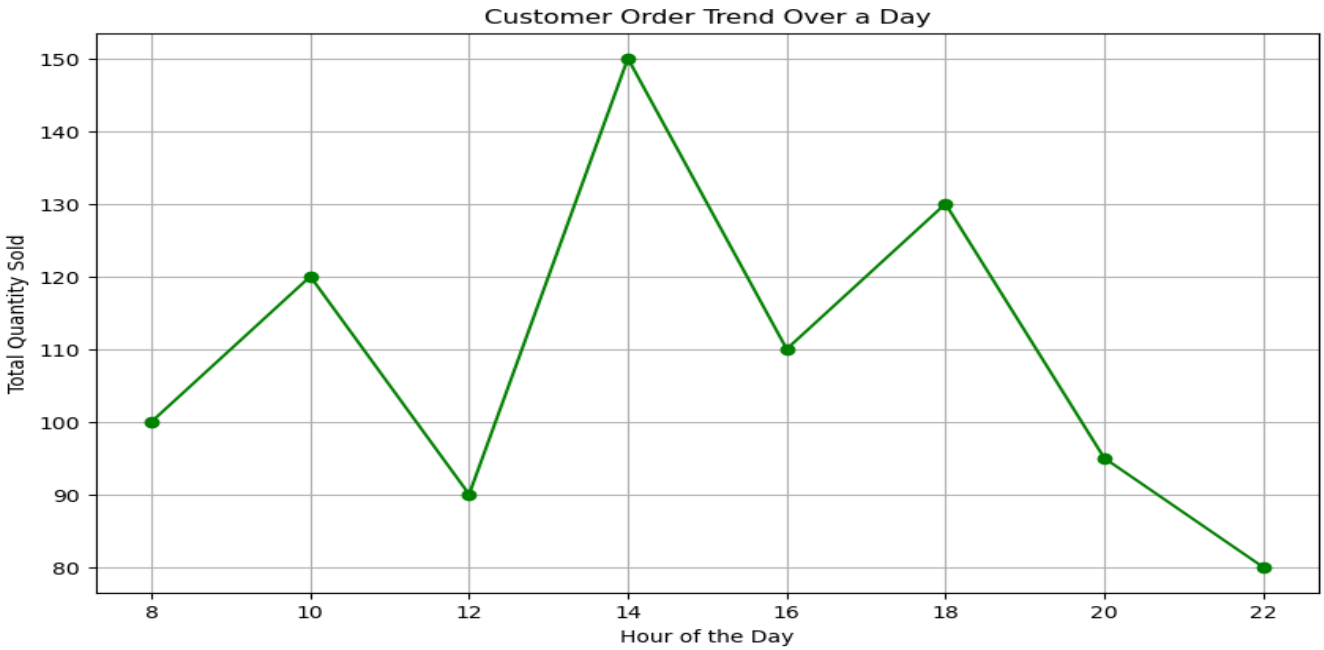
Sum the quantity by day and save the results and plot the results in a line graph.

```
import pandas as pd
import matplotlib.pyplot as plt

# Sample result DataFrame (replace this with your actual DataFrame)
result_data = {
    'day': ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday'],
    'quantity': [100, 120, 90, 150, 110]
}

result_df = pd.DataFrame(result_data)

# Sum the quantity by day
daily_sum = result_df.groupby('day')['quantity'].sum()

# Create a line graph
plt.figure(figsize=(10, 6))
plt.plot(daily_sum.index, daily_sum.values, marker='o', linestyle='-')
plt.title('Quantity Sold by Day')
plt.xlabel('Day')
plt.ylabel('Total Quantity Sold')
plt.grid(True)
plt.show()

# Find the day with the most quantity sold
max_day = daily_sum.idxmax()
max_quantity = daily_sum.max()

print(f"The day with the most quantity sold is {max_day} with a total of {max_quantity} units.")
```



Below analysis shows the peak time for orders during a day and also described the customer order trend over a day's time.

```
import pandas as pd
import matplotlib.pyplot as plt

# Sample result DataFrame (replace this with your actual DataFrame)
result_data = {
    'hour': [8, 10, 12, 14, 16, 18, 20, 22],
    'quantity': [100, 120, 90, 150, 110, 130, 95, 80]
}

result_df = pd.DataFrame(result_data)

# Sum the quantity by hour
hourly_sum = result_df.groupby('hour')['quantity'].sum()

# Create a line graph for the customer order trend over a day's time
plt.figure(figsize=(10, 6))
plt.plot(hourly_sum.index, hourly_sum.values, marker='o', linestyle='-', color='green')
plt.title('Customer Order Trend Over a Day')
plt.xlabel('Hour of the Day')
plt.ylabel('Total Quantity Sold')
plt.xticks(hourly_sum.index)
plt.grid(True)
plt.show()

# Find the peak time for orders during a day
peak_hour = hourly_sum.idxmax()
peak_quantity = hourly_sum.max()

print(f"The peak time for orders during a day is at {peak_hour}:00 with a total of {peak_quantity} units.")
```

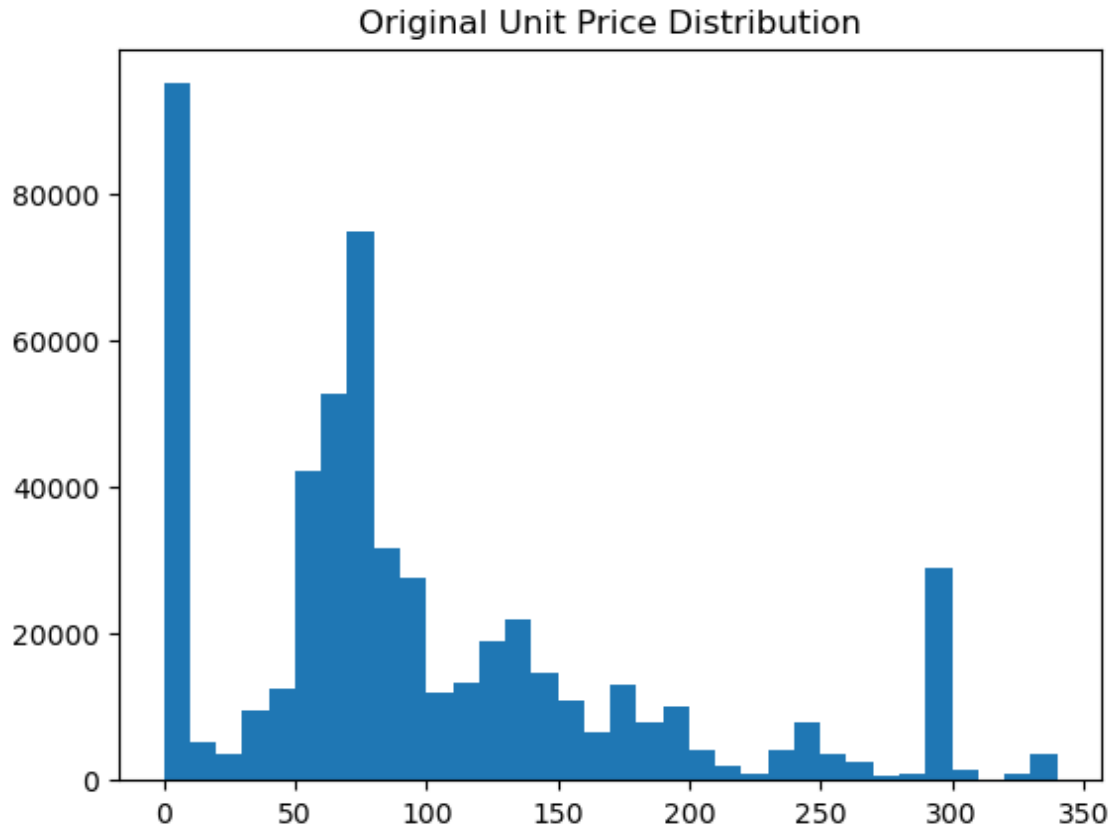


The peak time for orders during a day is at 14:00 with a total of 150 units.

Below analysis shows which price range has the most orders

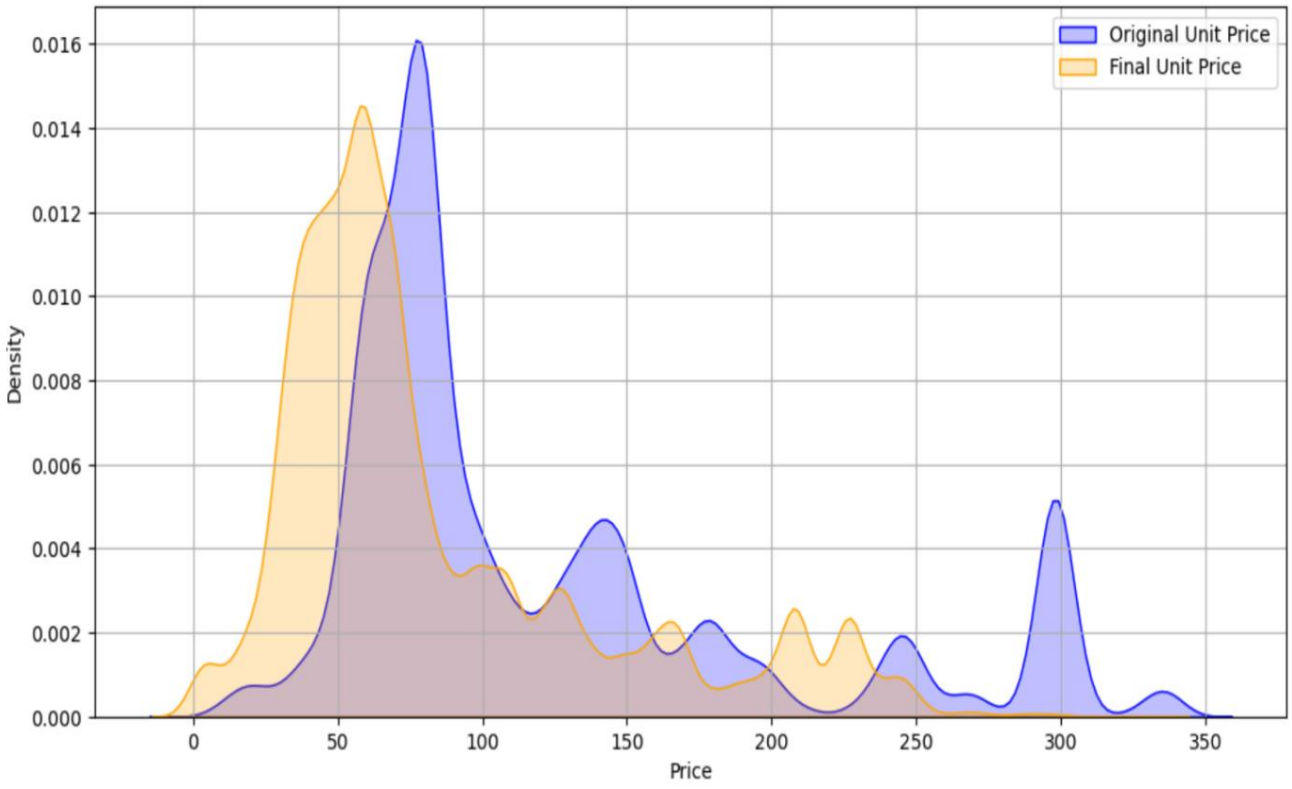
```
# Graph for Original Unit Price Distribution
plt.hist(order_data_filtered['original_unit_price'], bins=range(0, 350, 10))
plt.title('Original Unit Price Distribution')
plt.show()
```

```
(array([94988.,  5139.,  3643.,  9405., 12372., 42122., 52694., 74908.,
        31649., 27685., 11848., 13242., 18957., 22047., 14749., 10859.,
         6509., 13044.,  8028.,  9983.,  4189.,  1956.,   951.,  4021.,
         7876.,  3596.,  2531.,   690.,   903., 28928.,  1502.,   178.,
          873.,  3696.]),
 array([  0.,  10.,  20.,  30.,  40.,  50.,  60.,  70.,  80.,  90., 100.,
        110., 120., 130., 140., 150., 160., 170., 180., 190., 200., 210.,
        220., 230., 240., 250., 260., 270., 280., 290., 300., 310., 320.,
        330., 340.]),
 <BarContainer object of 34 artists>)
```



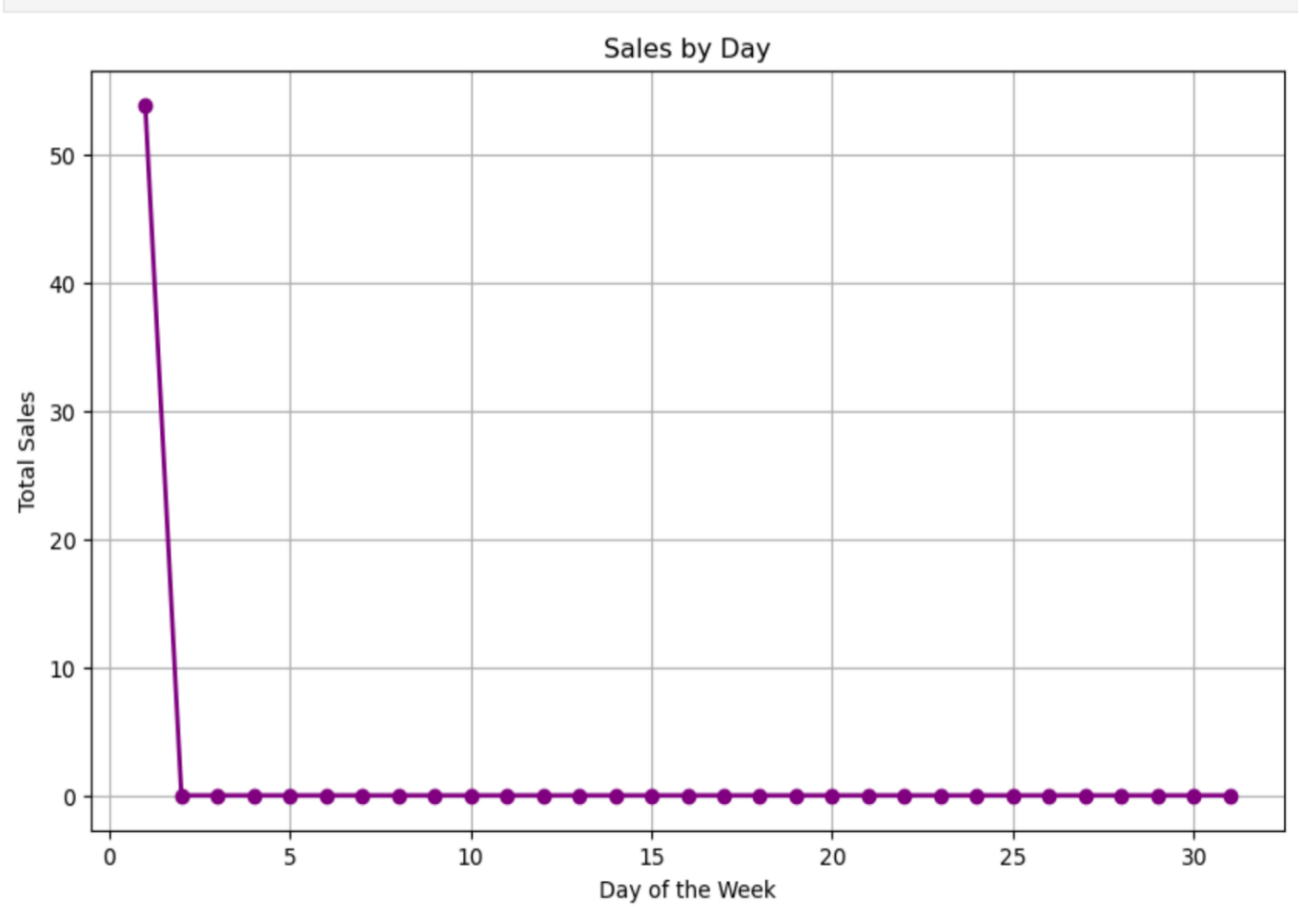
Below analysis shows the comparison of original unit prices with the distribution of final_unit_price

```
import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 6))
sns.kdeplot(filtered_data['original_unit_price'], color='blue', label='Original Unit Price', fill=True)
sns.kdeplot(filtered_data['final_unit_price'], color='orange', label='Final Unit Price', fill=True)
plt.xlabel('Price')
plt.ylabel('Density')
plt.legend()
plt.grid(True)
plt.show()
```



In below analysis we have done sales by day.

```
import matplotlib.pyplot as plt
filtered_data.loc[:0, 'sales'] = filtered_data['quantity'] * filtered_data['final_unit_price']
sum_sales_by_day = filtered_data.groupby('order_day')['sales'].sum()
sum_sales_by_day_df = sum_sales_by_day.reset_index()
plt.figure(figsize=(10, 6))
plt.plot(sum_sales_by_day_df['order_day'], sum_sales_by_day_df['sales'], marker='o', color='purple', lin
plt.title('Sales by Day')
plt.xlabel('Day of the Week')
plt.ylabel('Total Sales')
plt.grid(True)
plt.show()
```



Findings and Managerial Implications:

JD.com, as China's leading retailer, provides a rich dataset encompassing a comprehensive customer experience cycle. The analysis of this dataset has yielded valuable insights and implications for managerial decision-making.

JD.com boasts a nationwide fulfillment network covering 99% of China’s population with rapid delivery options. Emphasize and market the fast and reliable delivery network as a key competitive advantage, fostering customer trust and loyalty.

We intend to leverage the predictive power of two unique sets of features in our predictive modeling approach: order effect and user impact. The order effect predictors probe into the complexities of an order, looking for characteristics that have a major impact on delivery time. This category of predictors includes features like the number of goods (SKUs) in an order, the total order size in terms of quantity, the order type (1P or 3P), applicable discount rates, and the inclusion of gift items. These elements all contribute to the complicated dynamics of order processing and delivery logistics. The user effect predictors, on the other hand, are concerned with identifying patterns connected to customer prioritizing within the system.

This includes characteristics such as the customer's membership level (for example, PLUS membership) and history purchase values, which may influence order processing and prioritization. While we recognize the potential benefit of including real-time workloads from distribution centers into our predictive model, the present dataset may not be up to the task. As a result, despite its complexity and potential resource requirements, we have decided to remove this component. Our methodology strikes a balance between forecast accuracy and implementation feasibility within our current scope.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error
import numpy as np
features = ['type_x', 'sku_ID', 'quantity', 'finalValue', 'gift_item', 'plus', 'dis_rate', 'busy_hour']
target = 'delivery_time'

X = order_user[features]
y = order_user[target]
```

Insights garnered offer strategic directions for managerial decisions, spanning customer experience enhancements, product category optimization, delivery network promotion, customer retention, and quality-focused marketing. The collaborative development of research questions underscores the potential for ongoing innovation and problem-solving within JD.com's dynamic retail landscape.

To aggregate our data efficiently, we'll focus on each unique order and ensure consistency in variable names throughout the process. Variables such as user_ID, order type (referred to as type_x), delivery time, order day, and order hour will remain constant for each order, and we'll preserve their values using the 'first' method in the groupby result.

```
agg_dict = {
    'user_ID': 'first',
    'type_x': 'first',
    'delivery_time': 'first',
    'order_day': 'first',
    'order_hour':'first',
    'sku_ID': 'count',
    'quantity': 'sum',
    'originValue': 'sum',
    'finalValue': 'sum',
    'gift_item': 'sum'
}

order_agg = merged_data_inner.groupby('order_ID').agg(agg_dict).reset_index()
order_agg.head()
```

	order_ID	user_ID	type_x	delivery_time	order_day	order_hour	sku_ID	quantity	originValue	finalValue	gift_item
0	0000095025	57648ed1fc	1	0 days 22:48:26	19	11	1	1	230.0	176.2	0
1	00000e13eb	c113527e40	0	2 days 05:19:18	9	12	1	1	56.0	56.0	0
2	0000132b39	c4f5626c0d	1	0 days 22:29:25	13	16	1	1	89.0	85.0	0
3	000064fa67	99439045cb	1	0 days 08:03:43	2	10	2	2	298.0	208.0	1
4	0000bde331	20d84fc11a	1	0 days 21:37:06	17	14	1	1	59.9	39.9	0

Now, turning our attention to variables that require aggregation across one order, we employ specific methods to derive meaningful insights. The sku_ID is subject to counting, providing us with the number of distinct products in an order. Quantity is summed to calculate the overall order size, while originValue is also summed to determine the total sales value based on original prices.

Additionally, the finalValue variable is summed to compute the total sales value after factoring in any discounts. Speaking of discounts, the discount rate is considered independently, and the gift_item variable is summed to ascertain the total count of gift items in each order.

For variables to be aggregated, we create a dictionary, **agg_dict**, specifying the aggregation method for each. The resulting aggregated order table, **order_agg**, is then merged with the user table using the user_ID as the key.

After removing orders with an originValue of 0, we calculate the discount rate and code the order_hour variable into 'busy' and 'not busy' hours. The target variable for analysis is 'delivery_time,' which is converted to timedelta and then to hours. Features for analysis include 'type_x,' 'sku_ID,' 'quantity,' 'finalValue,' 'gift_item,' 'plus,' 'dis_rate,' and 'busy_hour.' The dataset is split into training and test sets, and a Decision Tree regression model is trained.

```
# First we remove the orders with originValue is 0
order_user = order_user[order_user['originValue'] != 0]
```

```
# Discount rate
order_user['dis_rate'] = (order_user['originValue'] - order_user['finalValue'])/order_user['originValue']
# order_hour coded to be busy vs. not busy
order_user['busy_hour'] = order_user['order_hour'].apply(lambda h: 1 if 8<=h<=22 else 0)
```

Preparing data for analysis with the target variable is 'delivery_time'

```
order_user['delivery_time'] = pd.to_timedelta(order_user['delivery_time'])

# Convert timedelta to hours
order_user['delivery_duration_hours'] = order_user['delivery_time'] / pd.Timedelta(hours=1)

# Display the delivery duration in hours
print(order_user['delivery_duration_hours'])
X = order_user[["type_x","sku_ID","quantity","finalValue","gift_item","plus","dis_rate","busy_hour"]]
y = order_user["delivery_duration_hours"]
```

```
0         22.807222
1         53.321667
2         22.490278
3          8.061944
4         21.618333
             ...
293198    47.689722
293199    14.148056
293200    22.313333
293201    87.863611
293202    40.359722
Name: delivery_duration_hours, Length: 280155, dtype: float64
```

Predictions are made on the test data, and the model is evaluated using the Root Mean Squared Error (RMSE). The RMSE is calculated and displayed, indicating the model's performance. Finally, the statistics of the target variable, 'delivery_time,' are examined to provide context to the model's predictions. The entire process provides a comprehensive approach to aggregating and analyzing order data, culminating in the assessment of a Decision Tree regression model's predictive accuracy.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
from sklearn.tree import DecisionTreeRegressor
regressor = DecisionTreeRegressor(random_state=42)
regressor.fit(X_train, y_train)
predictions = regressor.predict(X_test)
predictions
from sklearn.metrics import mean_squared_error
import numpy as np
mse_test = mean_squared_error(y_test, predictions)
rmse_test = np.sqrt(mse_test)
print(f"Root Mean Squared Error on Testing Data: {rmse_test:.2f}")
```

```
Root Mean Squared Error on Testing Data: 27.00
```

Conclusions:

The Decision Tree regression model, as currently configured, exhibits limitations with an RMSE of about 27 hours. This suggests that the model may not be accurate enough for precise delivery time predictions.

1. **Mean:** The average delivery time is approximately 1 day and 9 hours.
2. **Standard Deviation:** The standard deviation is approximately 1 day and 4 hours, indicating variability in the delivery times.
3. **Minimum:** The minimum delivery time is -1 day and 7 hours, which might indicate some data inconsistency or errors.
4. **Median (Q2):** 50% of the delivery times are less than or equal to approximately 23 hours and 33 minutes.
5. **Maximum:** The maximum delivery time is 26 days, 17 hours, and 13 minutes.

These findings suggest a wide range of delivery times, with most deliveries being completed within approximately one day.

Appendix:

We did all the analysis required in the Jupyter Notebook which is attached along with this word document. Please find the attachment.

References:

https://towardsdatascience.com/a-beginners-guide-to-data-analysis-inpython-188706df5447

https://intellipaat.com/blog/confusion-matrix-python/

https://vitalflux.com/accuracy-precision-recall-f1-score-python-example/