Retail-Delivery-Optimization-Case-Study

By Fatemeh Firouzi 11/20/2023

Introduction

Welcome to the business case analysis. This repository showcases my comprehensive case study analysis focused on optimizing delivery operations for a large retail chain. The project aims to enhance delivery efficiency, reduce costs, and improve customer satisfaction through data-driven insights."

Business Objectives

My primary objectives for this business case analysis are as follows:

1. **Delivery Cost Optimization:** Develop a mathematical delivery-cost equation to understand the factors influencing delivery cost and their respective coefficients.
2. **Warehouse Strategy:** Analyze the effectiveness of our three warehouses and propose strategies to enhance our delivery strategy, including warehouse placement and delivery radius.
3. **Customer Insights:** Gain actionable insights into customer distribution, preferences, and expedited delivery choices to tailor our services effectively.

Data and Tools

I will conduct the analysis using Python, leveraging the Pandas library for data manipulation and analysis. The dataset contains information about orders, customers, warehouses, and delivery-related metrics for 2019.

Methodology

I will follow a structured approach to address each objective, including data cleaning, mathematical modeling, and visualization. The analysis will lead to actionable recommendations for improving delivery services for Enterprise clients.

Let's embark on this journey of data-driven optimization!

In [1]:
```
# Import necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os
```

```
import warnings
import math
import ast

warnings.filterwarnings('ignore')
%matplotlib inline
```

In [2]:
```
# Load the Excel file
file_path = 'Data Workbook[76][66].xlsx'
xls = pd.ExcelFile(file_path)

# Read the two sheets into dataframes
sheet1 = xls.parse('dirty_data')
sheet2 = xls.parse('Missing Data')
sheet3 = xls.parse('Warehouse Info')
```

In [3]:
```
# Combine the two dataframes as needed
combined_df = pd.concat([sheet1, sheet2], axis=0)  # Concatenate vertically (along rows
```

In [4]:
```
# Display the first 5 rows
pd.options.display.max_columns = None
combined_df.head()
```

Out[4]:

	order_id	customer_id	date	nearest_warehouse	shopping_cart	order_price	delivery_charges
0	ORD010913	ID0361229947	2019-08-26 00:00:00	Bakers	[('Olivia x460', 1), ('Universe Note', 1), ('T...	14175.0	55.71
1	ORD353238	ID0214092261	2019-05-30 00:00:00	Bakers	[('Universe Note', 1), ('iAssist Line', 2), ('...	12810.0	84.86
2	ORD420507	ID0777791299	2019-04-15 00:00:00	Thompson	[('Candle Inferno', 1), ('Alcon 10', 1), ('pea...	22000.0	63.58
3	ORD322017	ID0591429491	2019-03-21 00:00:00	Nickolson	[('Toshika 750', 2), ('Candle Inferno', 1), ('...	19250.0	48.55
4	ORD289920	ID0702374154	2019-02-14 00:00:00	Thompson	[('Thunder line', 1), ('iStream', 2)]	2480.0	72.41

Data Preparation

In [5]:
```
print("The shape of the dataframe is:", combined_df.shape) # (n_instances, n_features)
```

```
The shape of the dataframe is: (1000, 16)
```

In [6]:
```
print("General Information of the data set: \n")
combined_df.info()
```

```
General Information of the data set:

<class 'pandas.core.frame.DataFrame'>
Int64Index: 1000 entries, 0 to 499
Data columns (total 16 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   order_id                       1000 non-null   object
 1   customer_id                    1000 non-null   object
 2   date                           1000 non-null   object
 3   nearest_warehouse              990 non-null    object
 4   shopping_cart                  1000 non-null   object
 5   order_price                    990 non-null    float64
 6   delivery_charges               1000 non-null   float64
 7   customer_lat                   990 non-null    float64
 8   customer_long                  990 non-null    float64
 9   coupon_discount                1000 non-null   int64
 10  order_total                    990 non-null    float64
 11  season                         990 non-null    object
 12  is_expedited_delivery          1000 non-null   bool
 13  distance_to_nearest_warehouse  990 non-null    float64
 14  latest_customer_review         1000 non-null   object
 15  is_happy_customer              990 non-null    float64
dtypes: bool(1), float64(7), int64(1), object(7)
memory usage: 126.0+ KB
```

In [7]:
```
print("The number of unique values per column: \n", combined_df.nunique())
# season need to be
```

```
The number of unique values per column:
 order_id                         1000
customer_id                        973
date                               349
nearest_warehouse                    5
shopping_cart                      877
order_price                        692
delivery_charges                   879
customer_lat                       964
customer_long                      964
coupon_discount                      5
order_total                        989
season                               8
is_expedited_delivery                2
distance_to_nearest_warehouse      937
latest_customer_review            1000
is_happy_customer                    2
dtype: int64
```

In [8]:
```
# Basic statistics summary
combined_df.describe().round(2)
```

Out[8]:

	order_price	delivery_charges	customer_lat	customer_long	coupon_discount	order_total	distanc
count	990.00	1000.00	990.00	990.00	1000.00	990.00	
mean	19432.18	77.17	-36.81	144.97	11.04	25712.48	
std	61860.81	14.53	8.61	0.02	8.69	195304.37	

	order_price	delivery_charges	customer_lat	customer_long	coupon_discount	order_total	distanc
min	580.00	46.20	-37.83	144.92	0.00	568.64	
25%	7095.00	66.38	-37.82	144.95	5.00	6477.46	
50%	12507.50	76.79	-37.81	144.96	10.00	11060.85	
75%	19422.50	83.87	-37.81	144.98	15.00	17366.90	
max	947691.00	114.04	37.83	145.02	25.00	5688269.60	

Data Cleaning

In [9]:
```
# Define a threshold for order_price outliers
outlier_threshold = 50000

# Remove rows with 'order_price' greater than the threshold
combined_df = combined_df[combined_df['order_price'] <= outlier_threshold]

# Fill missing values in 'order_price' column
combined_df['order_price'].fillna(
    (combined_df['order_total'] - combined_df['delivery_charges']) / (1 - combined_df['
    inplace=True
)

# Fill missing values in 'order_total' column
combined_df['order_total'].fillna(
    combined_df['order_price'] * (1 - combined_df['coupon_discount'] / 100) + combined_
    inplace=True
)
```

In [10]:
```
# Cleaning missing data
# Calculate and fill missing values in 'order_total' column based on the formula - we ha
#combined_df['order_total'].fillna(
#    combined_df['order_price'] * (1 - combined_df['coupon_discount'] / 100) +
#    combined_df['delivery_charges'], inplace=True)

# Calculate and replace all values in 'order_total' column based on the formula - fixed
#combined_df['order_total'] = (
#    combined_df['order_price'] * (1 - combined_df['coupon_discount'] / 100) +
#    combined_df['delivery_charges'] )
```

In [11]:
```
# Cleaning missing data
# Calculate and fill missing values in 'order_price' column based on the formula - we ha
#combined_df['order_price'].fillna((combined_df['order_total'] -
 #       combined_df['delivery_charges'])/ (1 - combined_df['coupon_discount'] / 100),
```

In [12]:
```
# Cleaning missing data
# Clean Date column: (MM/DD/YYYY format) for consistency
# We also need 'date' column to clean missing value for 'season' column

import pandas as pd
```

```
# Replace 'date' with the actual name of your date column if it's different
combined_df['date'] = pd.to_datetime(combined_df['date'], errors='coerce')

# Assuming the incorrect format is MM DD YYYY
combined_df['date'] = combined_df['date'].dt.strftime('%m/%d/%Y')

# Print the updated DataFrame
#print(combined_df)
# Cleaninig Season Column - formatting

# Assuming 'combined_df' is your DataFrame

# Clean the 'Date' column to ensure it's in the format (MM/DD/YYYY)
combined_df['date'] = pd.to_datetime(combined_df['date'], errors='coerce')

#The "season" column has 8 unique values, which may indicate the data needs to be cleane
# Define a mapping dictionary to convert seasons to lowercase and map to 4 unique values
season_mapping = {
    'winter': 'Winter',
    'autumn': 'Autumn',
    'summer': 'Summer',
    'spring': 'Spring'
}

# Use the map function to update the "season" column
combined_df['season'] = combined_df['season'].str.lower().map(season_mapping)

# Function to map a date to a season
def map_date_to_season(date):
    season_mapping = {
        (3, 1): 'Spring',   # Spring starts from March 1st
        (6, 1): 'Summer',   # Summer starts from June 1st
        (9, 1): 'Autumn',   # Autumn starts from September 1st
        (12, 1): 'Winter'   # Winter starts from December 1st
    }

    for (start_month, start_day), season in season_mapping.items():
        if (date.month == start_month and date.day >= start_day) or (date.month == start
            return season
    return None


# Fill null values in the 'season' column based on the 'Date' column
null_season_indices = combined_df['season'].isnull()
combined_df.loc[null_season_indices, 'season'] = combined_df.loc[null_season_indices, '

# Print the updated DataFrame
#print(combined_df)
```

In [13]:

```
# Cleaning missing data - 'distance_to_nearest_warehouse' column - Null Value

# Define a function to calculate the haversine distance between two points
def haversine(lat1, lon1, lat2, lon2):
    # Radius of the Earth in kilometers
    radius = 6371.0

    # Convert latitude and longitude from degrees to radians
    lat1 = math.radians(lat1)
```

```
    lon1 = math.radians(lon1)
    lat2 = math.radians(lat2)
    lon2 = math.radians(lon2)

    # Haversine formula
    dlon = lon2 - lon1
    dlat = lat2 - lat1
    a = math.sin(dlat/2)**2 + math.cos(lat1) * math.cos(lat2) * math.sin(dlon/2)**2
    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1-a))
    distance = radius * c

    return distance

# Iterate over rows in combined_df with null 'distance_to_nearest_warehouse' and calcul
for index, row in combined_df[combined_df['distance_to_nearest_warehouse'].isnull()].it
    customer_lat = row['customer_lat']
    customer_long = row['customer_long']
    nearest_warehouse = None
    min_distance = float('inf')

    # Calculate distance to each warehouse from sheet3
    for _, warehouse_row in sheet3.iterrows():
        warehouse_lat = warehouse_row['lat']
        warehouse_long = warehouse_row['lon']
        distance = haversine(customer_lat, customer_long, warehouse_lat, warehouse_long

        # Check if this warehouse is closer than the current nearest one
        if distance < min_distance:
            min_distance = distance
            nearest_warehouse = warehouse_row['names']

    # Update the 'distance_to_nearest_warehouse' column with the closest warehouse dist
    combined_df.at[index, 'distance_to_nearest_warehouse'] = min_distance
    combined_df.at[index, 'nearest_warehouse'] = nearest_warehouse

# Save the updated DataFrame to a new Excel file or overwrite the existing one
combined_df.to_excel('updated_file.xlsx', index=False)
```

In [14]:

```
# Check for null values in the 'distance_to_nearest_warehouse' column of the updated Da
null_values_check = combined_df['distance_to_nearest_warehouse'].isnull().sum()

if null_values_check == 0:
    print("No null values found in the 'distance_to_nearest_warehouse' column.")
else:
    print(f"{null_values_check} null values found in the 'distance_to_nearest_warehouse
```

```
No null values found in the 'distance_to_nearest_warehouse' column.
```

In [15]:

```
# drop a column that is extra and not used in this analysis
combined_df = combined_df.drop(columns = 'latest_customer_review')
```

In [16]:

```
# clean data - 'shopping_card' column - which converts all the text to lowercase.
combined_df['shopping_cart'] = combined_df['shopping_cart'].str.lower()
```

In [17]:

```python
import ast  # This module is used to safely evaluate the string as a literal

# Define a function to extract quantity from each tuple in the 'shopping_cart' column
def extract_quantity(cart_str):
    try:
        cart = ast.literal_eval(cart_str)  # Safely evaluate the string as a tuple
        quantities = [item[1] for item in cart]
    except (ValueError, SyntaxError):
        quantities = []  # Handle invalid or empty values gracefully
    return quantities

# Apply the extract_quantity function to create a new 'quantities' column
combined_df['quantities'] = combined_df['shopping_cart'].apply(extract_quantity)

# Display the updated DataFrame
#print(combined_df.head())
```

In [18]:

```python
# Define a function to extract quantity from each tuple in the 'shopping_cart' column
# In this code, we define a function 'extract_quantity' that parses the 'shopping_cart'
# extracts item quantities, and converts item names to lowercase for consistency.
def extract_quantity(cart_str):
    try:
        cart = ast.literal_eval(cart_str)  # Safely evaluate the string as a tuple
        quantities = {item[0].lower(): item[1] for item in cart}  # Convert item names
    except (ValueError, SyntaxError):
        quantities = {}  # Handle invalid or empty values gracefully
    return quantities

# Apply the extract_quantity function to create a new 'item_quantities' column
combined_df['item_quantities'] = combined_df['shopping_cart'].apply(extract_quantity)

# Expand the 'item_quantities' dictionary column into separate columns
combined_df = pd.concat([combined_df, combined_df['item_quantities'].apply(pd.Series)],

# Replace NaN values with zero for the new item columns
combined_df = combined_df.fillna(0)

# combined_df['total_quantity'] = combined_df.iloc[:, -len(combined_df['item_quantities

# Note:
# We then apply this function to create a new 'item_quantities' column, expanding it in
# NaN values are replaced with zero for the new item columns to handle missing data.

# The code helps prepare the data for analysis by providing a clear breakdown of item q
# This information can be useful for various analytical tasks involving order items and

# Display the updated DataFrame
print(combined_df.head())
```

```
    order_id   customer_id       date nearest_warehouse  \
0  ORD010913  ID0361229947 2019-08-26            Bakers
1  ORD353238  ID0214092261 2019-05-30            Bakers
2  ORD420507  ID0777791299 2019-04-15          Thompson
3  ORD322017  ID0591429491 2019-03-21         Nickolson
4  ORD289920  ID0702374154 2019-02-14          Thompson

                                       shopping_cart  order_price  \
0  [('olivia x460', 1), ('universe note', 1), ('t...      14175.0
```

```
1  [('universe note', 1), ('iassist line', 2), ('...      12810.0
2  [('candle inferno', 1), ('alcon 10', 1), ('pea...      22000.0
3  [('toshika 750', 2), ('candle inferno', 1), ('...      19250.0
4             [('thunder line', 1), ('istream', 2)]       2480.0

   delivery_charges  customer_lat  customer_long  coupon_discount  \
0             55.71    -37.800372     144.975276                0
1             84.86    -37.809083     145.011628               10
2             63.58    -37.808332     144.954755                5
3             48.55    -37.818227     144.965114               10
4             72.41    -37.807499     144.942120               25

   order_total  season  is_expedited_delivery  distance_to_nearest_warehouse  \
0     14230.71  Winter                  False                        94.9734
1     11613.86  Autumn                   True                        76.3419
2     20963.58  Autumn                  False                        73.8324
3     17373.55  Autumn                  False                        68.8892
4      1932.41  Summer                  False                        66.8512

   is_happy_customer    quantities  \
0                0.0  [1, 1, 2, 2]
1                1.0  [1, 2, 2, 2]
2                1.0     [1, 1, 2]
3                0.0  [2, 1, 1, 1]
4                1.0        [1, 2]

                                  item_quantities  olivia x460  \
0  {'olivia x460': 1, 'universe note': 1, 'toshik...          1.0
1  {'universe note': 1, 'iassist line': 2, 'lucen...          2.0
2  {'candle inferno': 1, 'alcon 10': 1, 'peartv': 2}          0.0
3  {'toshika 750': 2, 'candle inferno': 1, 'alcon...          0.0
4                 {'thunder line': 1, 'istream': 2}          0.0

   universe note  toshika 750  candle inferno  iassist line  lucent 330s  \
0            1.0          2.0             2.0           0.0          0.0
1            1.0          0.0             0.0           2.0          2.0
2            0.0          0.0             1.0           0.0          0.0
3            0.0          2.0             1.0           0.0          1.0
4            0.0          0.0             0.0           0.0          0.0

   alcon 10  peartv  thunder line  istream
0       0.0     0.0           0.0      0.0
1       0.0     0.0           0.0      0.0
2       1.0     2.0           0.0      0.0
3       1.0     0.0           0.0      0.0
4       0.0     0.0           1.0      2.0
```

In [19]:

```
# Data is cleaned now
print("General Information of the data set: \n")
combined_df.info()
```

```
General Information of the data set:

<class 'pandas.core.frame.DataFrame'>
Int64Index: 975 entries, 0 to 499
Data columns (total 27 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   order_id                      975 non-null    object
 1   customer_id                   975 non-null    object
 2   date                          975 non-null    datetime64[ns]
 3   nearest_warehouse             975 non-null    object
 4   shopping_cart                 975 non-null    object
 5   order_price                   975 non-null    float64
```

```
 6   delivery_charges               975 non-null    float64
 7   customer_lat                   975 non-null    float64
 8   customer_long                  975 non-null    float64
 9   coupon_discount                975 non-null    int64
 10  order_total                    975 non-null    float64
 11  season                         975 non-null    object
 12  is_expedited_delivery          975 non-null    bool
 13  distance_to_nearest_warehouse  975 non-null    float64
 14  is_happy_customer              975 non-null    float64
 15  quantities                     975 non-null    object
 16  item_quantities                975 non-null    object
 17  olivia x460                    975 non-null    float64
 18  universe note                  975 non-null    float64
 19  toshika 750                    975 non-null    float64
 20  candle inferno                 975 non-null    float64
 21  iassist line                   975 non-null    float64
 22  lucent 330s                    975 non-null    float64
 23  alcon 10                       975 non-null    float64
 24  peartv                         975 non-null    float64
 25  thunder line                   975 non-null    float64
 26  istream                        975 non-null    float64
dtypes: bool(1), datetime64[ns](1), float64(17), int64(1), object(7)
memory usage: 222.8+ KB
```

A - The Mathematical Delivery-Cost Equation

In [20]:

```
# Data preparation before starting this analysis
# Perform one-hot encoding for the "season" column
combined_df = pd.get_dummies(combined_df, columns=['season'], prefix='season')

# Now, the DataFrame will have columns like 'season_Spring', 'season_Summer', 'season_Fa
# These columns will have binary values (0 or 1) indicating the presence of each season
```

In [21]:

```
# Perform one-hot encoding for the "is_expedited_delivery" column
combined_df = pd.get_dummies(combined_df, columns=['is_expedited_delivery'], prefix='ex

# Now, combined_df will have columns like 'expedited_True' and 'expedited_False'
# These columns will have binary values (0 or 1) indicating whether expedited delivery
```

In [22]:

```
#The code provided defines a function calculate_delivery_cost that calculates the delive
#based on several factors and their respective coefficients.

import pandas as pd

# Load your DataFrame, assuming it's named 'combined_df'
# combined_df = pd.read_csv('your_data.csv')  # Load your data

# Define coefficients for factors affecting delivery cost
coefficient_distance = 0.2  # Example coefficient for distance
coefficient_expedited = 1.5  # Example coefficient for expedited delivery
coefficient_coupon = -0.1  # Example coefficient for coupon discount (negative since it
coefficient_delivery = 1.0  # Example coefficient for delivery charges

# Define coefficients for season factors
coefficient_spring = 0.8  # Example coefficient for Spring
coefficient_summer = 1.2  # Example coefficient for Summer (higher due to potential inc
coefficient_autumn = 1.0  # Example coefficient for Autumn
```

```
coefficient_winter = 0.9  # Example coefficient for Winter

# Calculate the delivery cost for each row
def calculate_delivery_cost(row):
    delivery_cost = (
        coefficient_distance * row['distance_to_nearest_warehouse'] +
        coefficient_expedited * row['expedited_True'] +
        coefficient_coupon * row['coupon_discount'] +
        coefficient_delivery * row['delivery_charges'] +
        coefficient_spring * (1 if row['season_Spring'] == 'Spring' else 0) +
        coefficient_summer * (1 if row['season_Summer'] == 'Summer' else 0) +
        coefficient_autumn * (1 if row['season_Autumn'] == 'Autumn' else 0) +
        coefficient_winter * (1 if row['season_Winter'] == 'Winter' else 0)
    )
    return delivery_cost

# Calculate delivery cost for each row and add a new column 'delivery_cost'
combined_df['delivery_cost'] = combined_df.apply(calculate_delivery_cost, axis=1)

# Print the updated DataFrame with delivery cost
print(combined_df[['order_id', 'delivery_cost']])
```

```
      order_id  delivery_cost
0    ORD010913       74.70468
1    ORD353238      100.62838
2    ORD420507       77.84648
3    ORD322017       61.32784
4    ORD289920       83.28024
..         ...            ...
495  ORD289820       44.76208
496  ORD425999       81.89020
497  ORD252675       77.28126
498  ORD215989      105.38578
499  ORD414852       75.66292

[975 rows x 2 columns]
```

In [23]:

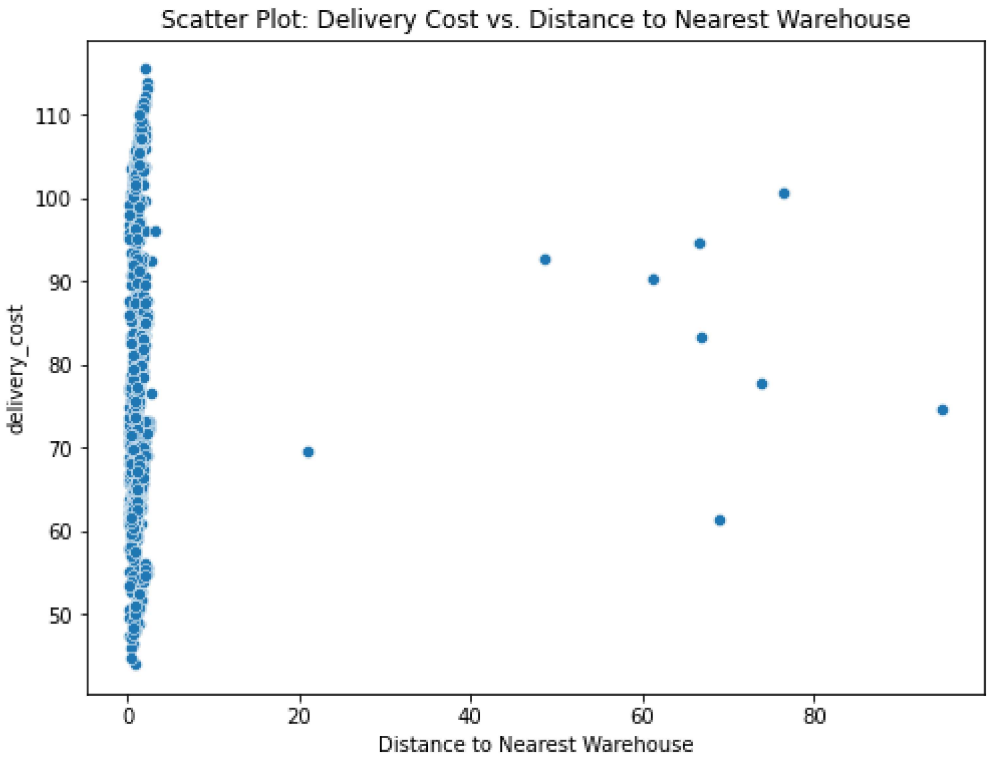
```
# Correlation Analysis
# Calculate correlation matrix
correlation_matrix = combined_df.corr()

# Correlation with delivery cost
delivery_cost_correlation = correlation_matrix['delivery_cost'].sort_values(ascending=F
print(delivery_cost_correlation)
#factors like delivery_charges, expedited_True (choosing expedited delivery), and certa
#tend to increase the delivery cost, while factors like season_Winter, expedited_False
#and certain items (e.g., toshika 750) tend to decrease the delivery cost.
```

```
delivery_cost                  1.000000
delivery_charges               0.993756
expedited_True                 0.643525
season_Spring                  0.445562
is_happy_customer              0.390348
season_Summer                  0.100013
lucent 330s                    0.075807
season_0                       0.062223
distance_to_nearest_warehouse  0.054314
olivia x460                    0.051387
peartv                         0.015436
customer_long                  0.013819
customer_lat                   0.006849
toshika 750                   -0.011229
```

```
order_total                    -0.014519
candle inferno                 -0.017386
alcon 10                       -0.025640
order_price                    -0.029631
iassist line                   -0.037439
universe note                  -0.038173
thunder line                   -0.038346
istream                        -0.039941
coupon_discount                -0.042127
season_Autumn                  -0.270649
season_Winter                  -0.299822
expedited_False                -0.643525
Name: delivery_cost, dtype: float64
```

In [24]:

```python
# Data Visualization
# Scatter plot between delivery cost and distance to nearest warehouse
plt.figure(figsize=(8, 6))
sns.scatterplot(data=combined_df, x='distance_to_nearest_warehouse', y='delivery_cost')
plt.title('Scatter Plot: Delivery Cost vs. Distance to Nearest Warehouse')
plt.xlabel('Distance to Nearest Warehouse')
plt.ylabel('delivery_cost')
plt.show()
```



In [25]:

```python
# clean outleirs for the 'distance_to_nearest_warehouse' column

# Define a function to calculate the haversine distance between two points
def haversine(lat1, lon1, lat2, lon2):
    # Radius of the Earth in kilometers
    radius = 6371.0

    # Convert latitude and longitude from degrees to radians
    lat1 = math.radians(lat1)
    lon1 = math.radians(lon1)
    lat2 = math.radians(lat2)
    lon2 = math.radians(lon2)
```

```
    # Haversine formula
    dlon = lon2 - lon1
    dlat = lat2 - lat1
    a = math.sin(dlat/2)**2 + math.cos(lat1) * math.cos(lat2) * math.sin(dlon/2)**2
    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1-a))
    distance = radius * c

    return distance

# Set the maximum allowed distance for cleaning outliers (5 miles)
max_allowed_distance = 5.0

# Iterate over rows in combined_df with 'distance_to_nearest_warehouse' greater than ma
for index, row in combined_df[combined_df['distance_to_nearest_warehouse'] > max_allowe
    customer_lat = row['customer_lat']
    customer_long = row['customer_long']
    nearest_warehouse = None
    min_distance = float('inf')

    # Calculate distance to each warehouse from sheet3
    for _, warehouse_row in sheet3.iterrows():
        warehouse_lat = warehouse_row['lat']
        warehouse_long = warehouse_row['lon']
        distance = haversine(customer_lat, customer_long, warehouse_lat, warehouse_long

        # Check if this warehouse is closer than the current nearest one
        if distance < min_distance:
            min_distance = distance
            nearest_warehouse = warehouse_row['names']

    # Update the 'distance_to_nearest_warehouse' column with the closest warehouse dist
    combined_df.at[index, 'distance_to_nearest_warehouse'] = min_distance
    combined_df.at[index, 'nearest_warehouse'] = nearest_warehouse

# Verify that the outliers have been replaced
print(combined_df['distance_to_nearest_warehouse'].describe())
```

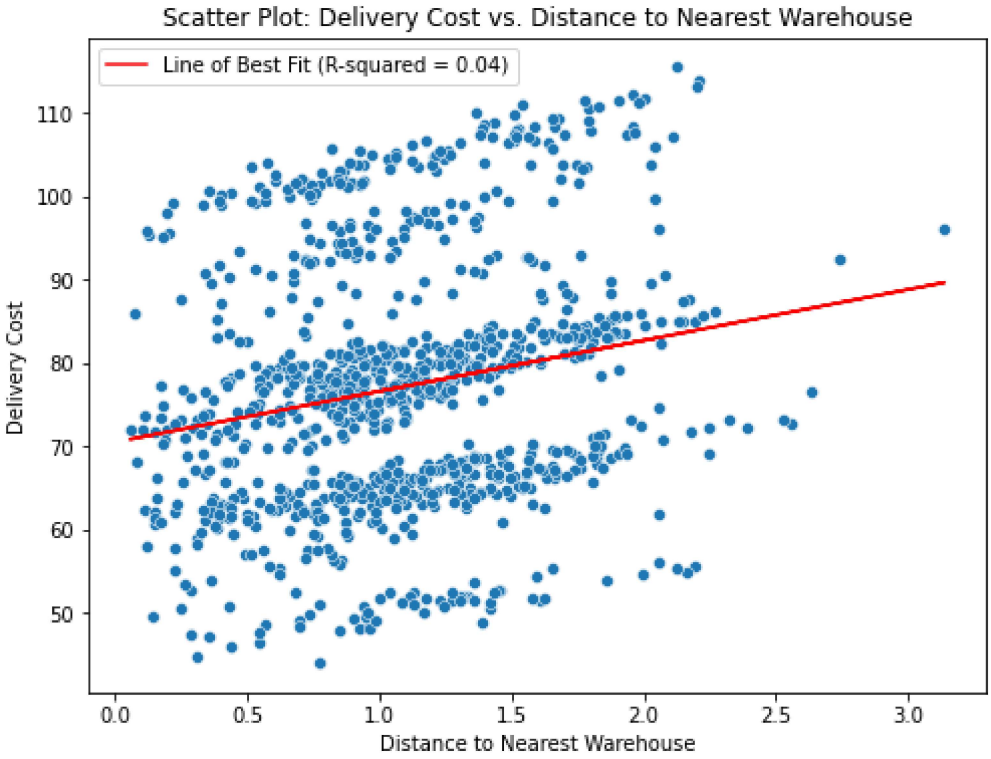
```
count    975.000000
mean       1.076821
std        0.492492
min        0.054900
25%        0.741000
50%        1.037300
75%        1.396950
max        3.138800
Name: distance_to_nearest_warehouse, dtype: float64
```

In [26]:

```
# Data Visualization
# Scatter plot between delivery cost and distance to nearest warehouse - rerun after cl
from scipy import stats

plt.figure(figsize=(8, 6))
sns.scatterplot(data=combined_df, x='distance_to_nearest_warehouse', y='delivery_cost')
plt.title('Scatter Plot: Delivery Cost vs. Distance to Nearest Warehouse')
plt.xlabel('Distance to Nearest Warehouse')
plt.ylabel('Delivery Cost')

# Perform linear regression to calculate the line equation
slope, intercept, r_value, p_value, std_err = stats.linregress(combined_df['distance_to
```

```
# Plot the line
x_values = combined_df['distance_to_nearest_warehouse']
y_values = slope * x_values + intercept
plt.plot(x_values, y_values, color='red', label=f'Line of Best Fit (R-squared = {r_valu

# Add a legend
plt.legend()

# Print R-squared, p-value, and standard error
print(f"R-squared: {r_value**2:.2f}")
print(f"P-value: {p_value:.4f}")
print(f"Standard Error: {std_err:.4f}")

plt.show()

# Interpretation:
# The scatter plot shows a relationship between delivery cost and the distance to the n
# It can be observed that as the distance to the nearest warehouse increases, delivery
# This suggests that distance to the nearest warehouse is not a very strong factor infl

# Regression Analysis:
# R-squared: 0.04 - This indicates that only 4% of the variance in delivery cost can be
# P-value: 0.0000 - The p-value is very small, indicating that the relationship between
# Standard Error: 0.9122 - The standard error of the estimate is relatively high, sugge
```

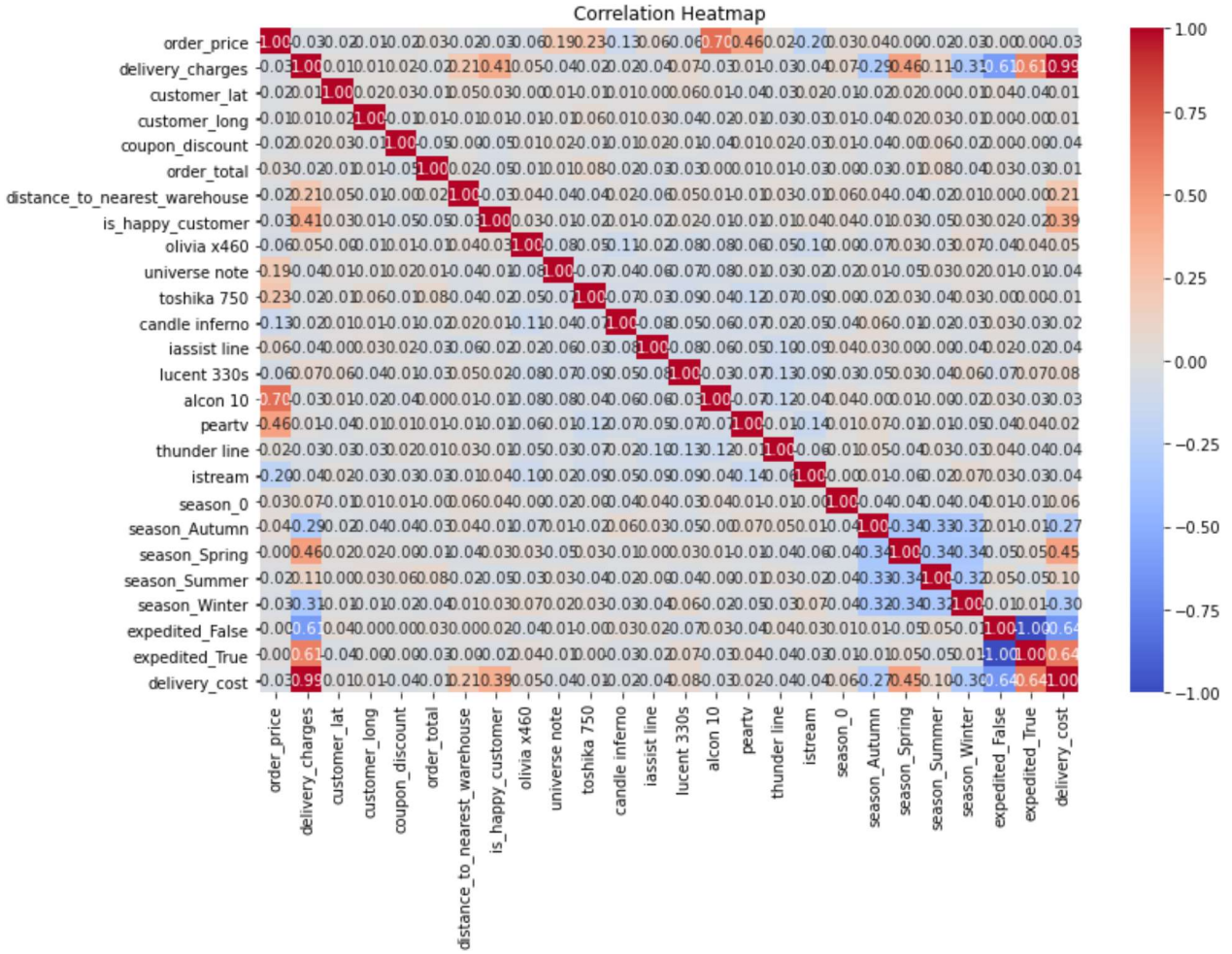
```
R-squared: 0.04
P-value: 0.0000
Standard Error: 0.9264
```



In [27]:

```
# Calculate the correlation matrix
correlation_matrix = combined_df.corr()

# Create a heatmap for an overall visualization
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
```

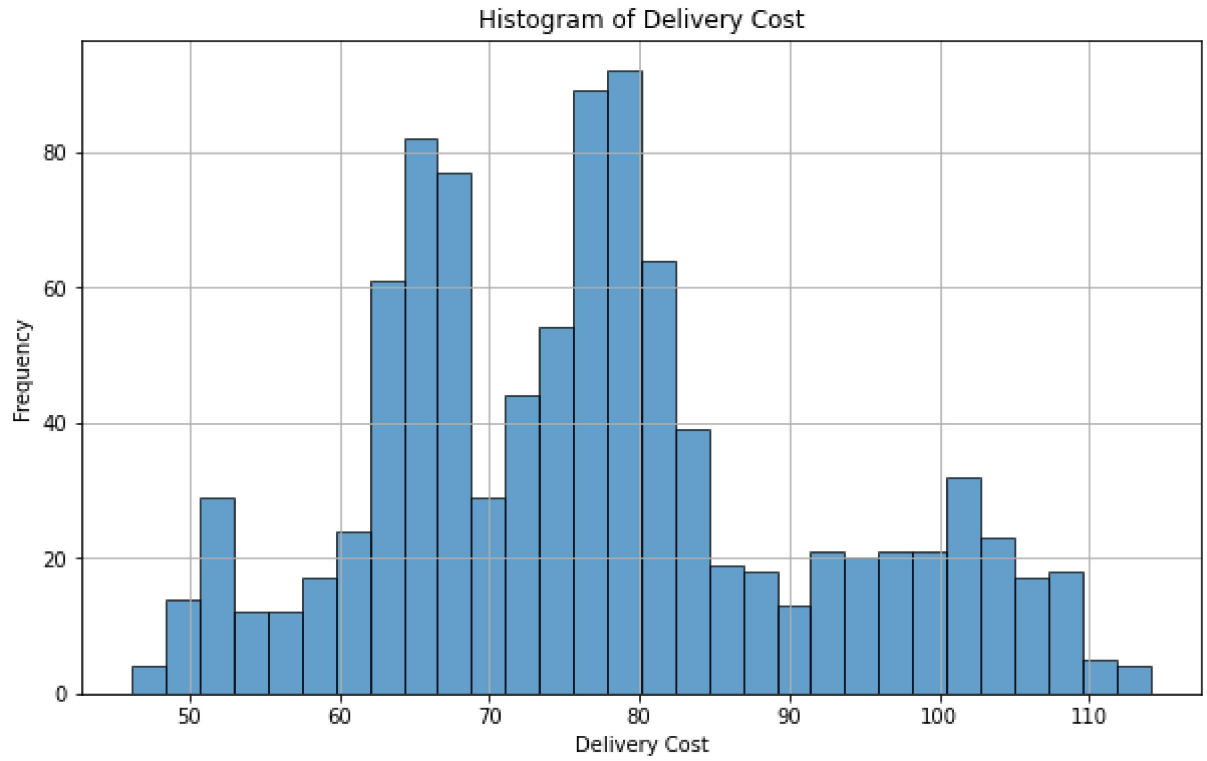
```
plt.title('Correlation Heatmap')
plt.show()
# Heatmap shows delivery_cost has a positive correlation with expefite_delivery (expedi
# also season and happy customer factors.
```

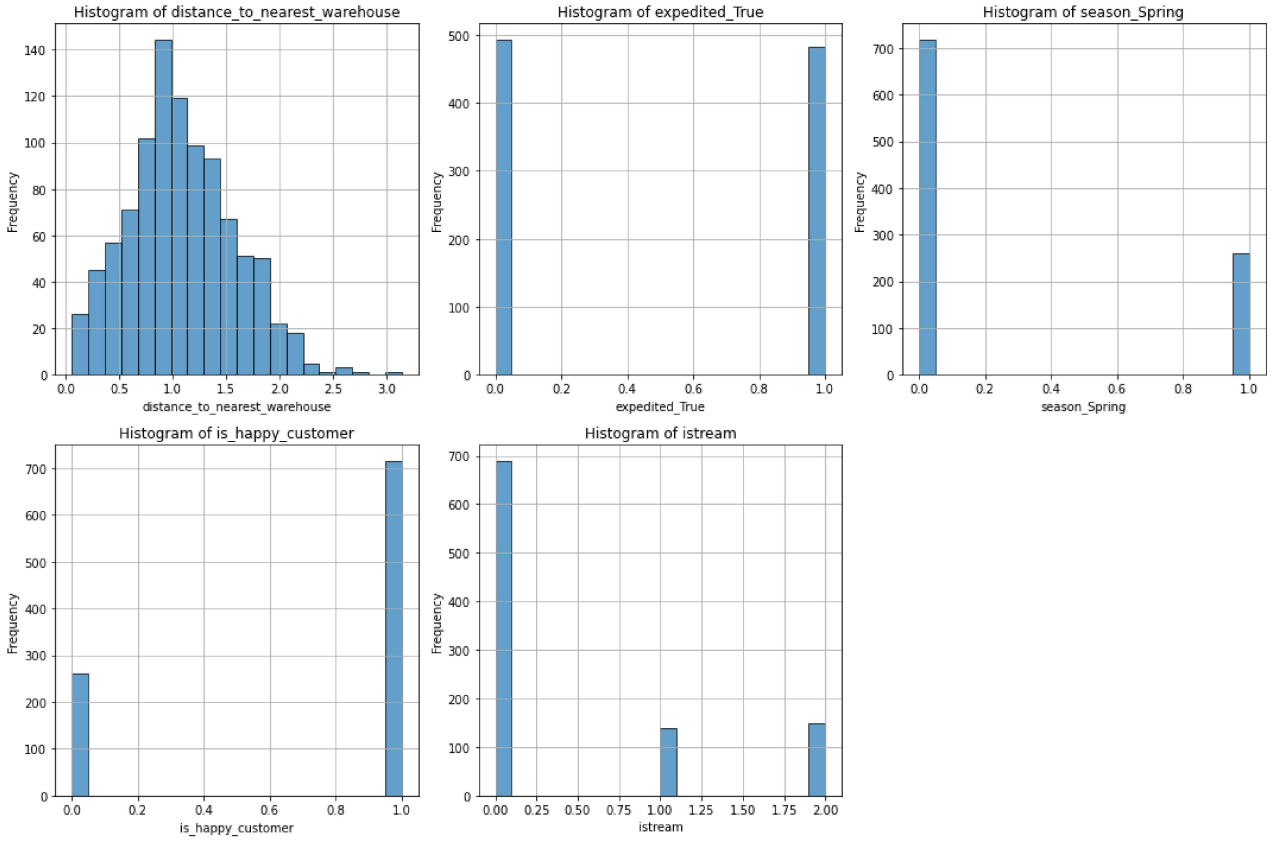


In [28]:

```
import matplotlib.pyplot as plt

# Create a histogram for delivery cost
plt.figure(figsize=(10, 6))
plt.hist(combined_df['delivery_charges'], bins=30, edgecolor='k', alpha=0.7)
plt.title('Histogram of Delivery Cost')
plt.xlabel('Delivery Cost')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()

# Create histograms for individual factors (e.g., distance_to_nearest_warehouse)
factor_names = ['distance_to_nearest_warehouse', 'expedited_True', 'season_Spring', 'is_
plt.figure(figsize=(15, 10))
for i, factor in enumerate(factor_names):
    plt.subplot(2, 3, i + 1)
    plt.hist(combined_df[factor], bins=20, edgecolor='k', alpha=0.7)
    plt.title(f'Histogram of {factor}')
    plt.xlabel(factor)
    plt.ylabel('Frequency')
    plt.grid(True)
```

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





Analysis of the histograms provides valuable insights into the distribution of data in the dataset. Here's a summary of my observations and insights:

Distance to Nearest Warehouse Histogram:

The histogram for "distance_to_nearest_warehouse" is positively skewed, suggests that the majority of deliveries are close to a warehouse, resulting in lower delivery costs. However, there are a few deliveries that are far away from a warehouse, which could be driving up the mean delivery cost. Addressing the outliers in terms of delivery distance may help in optimizing delivery routes and reducing costs.

Happy Customer Histogram:

The histogram for "is_happy_customer" indicates a high frequency of happy customers. More than 70% of customers are categorized as happy, which is a positive sign for customer satisfaction.

Season (Spring) Histogram:

The histogram for "season_Spring" reveals a considerable number of orders during the spring season. Close to 300 out of 1000 orders were placed during the spring. This insight can be useful for demand forecasting and ensuring that sufficient inventory and resources are available during peak seasons.

Expedited Delivery Histogram:

The histogram for "expedited_True" shows that almost 50% of orders requested expedited delivery. Offering expedited delivery options is popular among customers and can be a competitive advantage.

Quantity Frequencies for Order Items:

Considering the extraction of order items and their quantities from the "shopping_cart" column, and analyzing the frequencies of different order items can also provide insights into popular products and customer preferences. This information can be valuable for inventory management and marketing strategies.

Linear Regression Analysis

In [29]:

```
# Linear Regression Coefficients Analysis for Delivery Charges Prediction
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
# Drop any rows with null values
combined_df.dropna(axis=0, how='any', inplace=True)


x = combined_df[['distance_to_nearest_warehouse', 'expedited_True', 'coupon_discount',
                 'season_Autumn', 'season_Winter', 'olivia x460', 'universe note', 'tosh
                , 'lucent 330s', 'alcon 10', 'peartv', 'thunder line', 'istream' ]]
y = combined_df['delivery_charges']
```

```
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=1

from sklearn.linear_model import LinearRegression
model = LinearRegression(fit_intercept=False)
model.fit(x_train, y_train)
print('Coefficients: \n', model.coef_)

# The coefficients of a linear regression model represent the relationship between each
# In this context, we're analyzing the impact of various factors on delivery charges.
# Key findings from the coefficients:

# Expedited delivery significantly increases delivery charges.
# Seasons, especially Spring, have a substantial impact on charges.
# Certain products also affect charges differently.
# Distance to the nearest warehouse has a moderate impact.
# Coupon discounts have a minor negative influence.
```

```
Coefficients:
 [9.45542518e+00 1.86240016e+01 5.64079930e-02 6.38187888e+01
 5.76107434e+01 4.60200750e+01 4.50839248e+01 1.02183359e+00
 1.80313162e-01 7.90669096e-01 6.37603751e-01 4.43417428e-01
 9.81645486e-01 5.79662386e-01 1.17110872e+00 4.67386273e-01
 1.47611338e+00]
```

In [30]:

```
# To make Coefficients visible clearly
params = pd.Series(model.coef_, index=x.columns)
params
```

Out[30]:

```
distance_to_nearest_warehouse     9.455425
expedited_True                   18.624002
coupon_discount                   0.056408
season_Spring                    63.818789
season_Summer                    57.610743
season_Autumn                    46.020075
season_Winter                    45.083925
olivia x460                       1.021834
universe note                     0.180313
toshika 750                       0.790669
candle inferno                    0.637604
iassist line                      0.443417
lucent 330s                       0.981645
alcon 10                          0.579662
peartv                            1.171109
thunder line                      0.467386
istream                           1.476113
dtype: float64
```

Analysis of delivery cost-equation:

Expedited delivery:

It has a numerical coefficient of 18.62. Expedited delivery is a significant factor in increasing the delivery-cost. This is also understandable as priortizing the delivery could result in more labor cost and direct deliveries.

Distance to warehouse

It has a numerical coefficient of 9.45. It implies that the distance to warehouse increases the delivery cost as covering more distance requires more fuel and more labor cost.

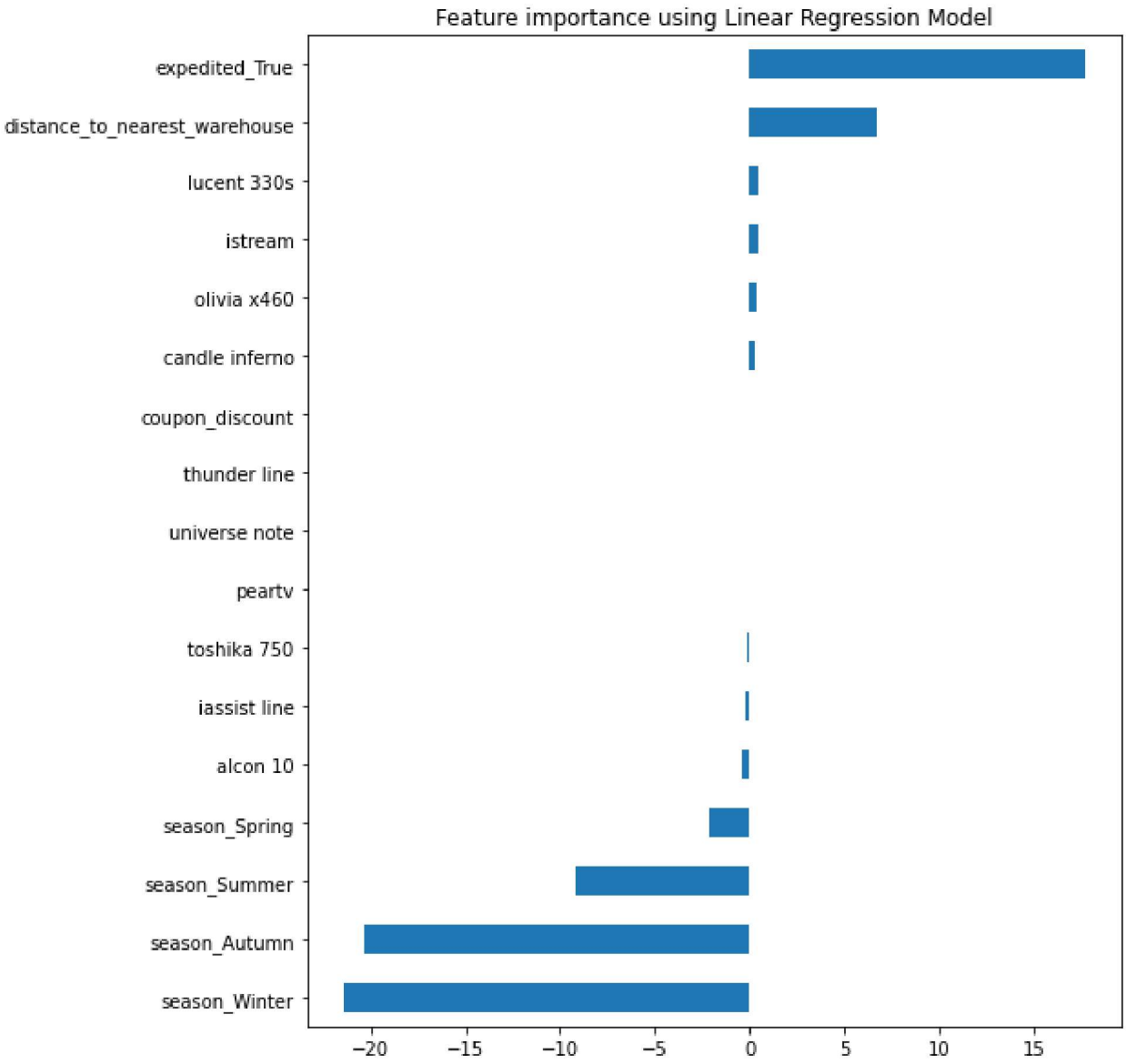
Seasons

Spring seasona has a numerical coefficial close to 64. It implies the positive impact of this season in our analysis. We may need to deep dive more to find out what could be the reason for that.

In [31]:
```
# Analyzing Feature Importance using Linear Regression
# In this code, we're using a Linear Regression model to determine the importance of ea
#the target variable.
from sklearn.linear_model import LinearRegression

# Create a Linear Regression model
model_linear_reg = LinearRegression()
model_linear_reg.fit(x, y)

# Get feature importance coefficients
coef = pd.Series(model_linear_reg.coef_, index=x.columns)
print("Linear Regression picked " + str(sum(coef != 0)) +
      " variables and eliminated the other " + str(sum(coef == 0)) + " variables")

# Sort and plot feature importance
imp_coef = coef.sort_values()
plt.rcParams['figure.figsize'] = (8.0, 10.0)
imp_coef.plot(kind="barh")
plt.title("Feature importance using Linear Regression Model")

# Key findings from the analysis:

# All 17 variables are considered important by the Linear Regression model, and none ha
# The importance of each feature is reflected in the coefficient values, which can be p
# The plot provides a visual representation of feature importance.
```

```
Linear Regression picked 17 variables and eliminated the other 0 variables
```

Out[31]:
```
Text(0.5, 1.0, 'Feature importance using Linear Regression Model')
```



In [32]:

```
# To remove less important factors in the analysis, I perform feature selection
# Define a threshold for feature importance
threshold = 0.05  # You can adjust this threshold as needed

# Filter the features based on the threshold
selected_features = coef[coef.abs() > threshold]

# Print the selected features
print("Selected Features:")
print(selected_features)

# Create a new DataFrame with only the selected features
selected_df = x[selected_features.index]

# Now, 'selected_df' contains only the selected important features
```

```
Selected Features:
distance_to_nearest_warehouse     6.732012
expedited_True                   17.647701
season_Spring                    -2.171492
season_Summer                    -9.195909
season_Autumn                   -20.315341
```

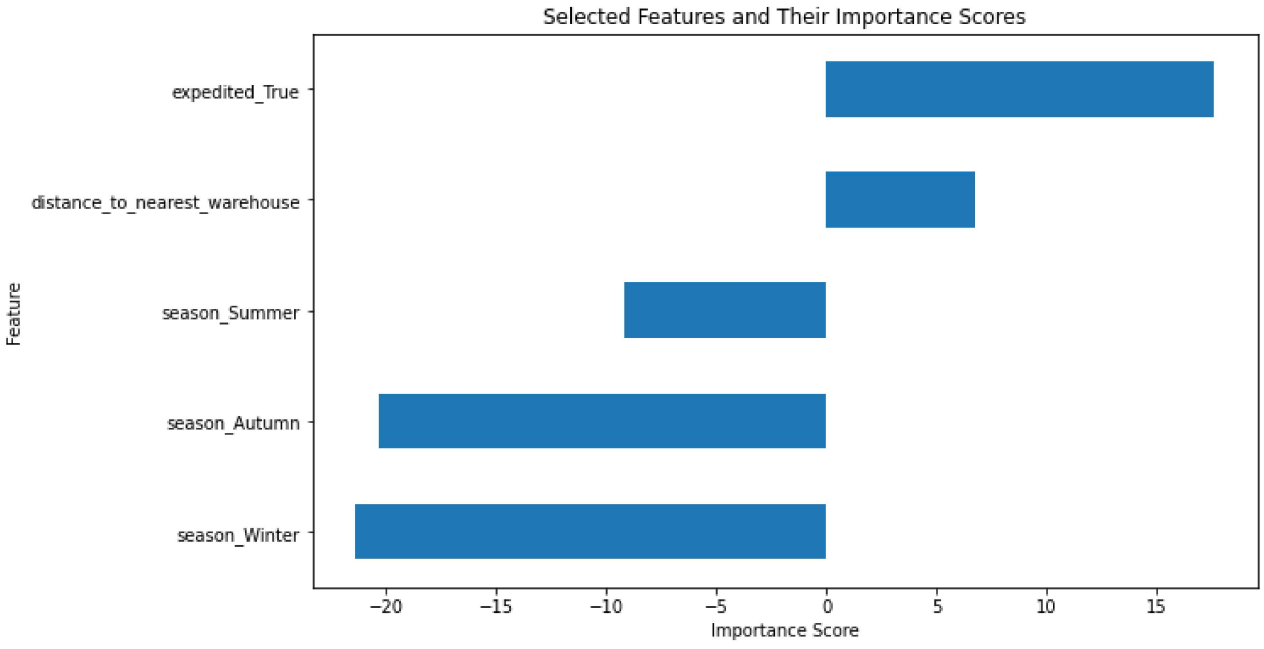
```
season_Winter                -21.401128
olivia x460                    0.373063
toshika 750                   -0.128173
candle inferno                 0.233248
iassist line                  -0.277743
lucent 330s                    0.439064
alcon 10                      -0.443139
istream                        0.419215
dtype: float64
```

In [33]:

```python
# Define a threshold for feature importance ( this threshold can be adjusted this thresh
threshold = 5.0

# Select features with importance scores above the threshold
selected_features = coef[coef.abs() > threshold].index.tolist()

# Create the selected DataFrame
selected_df = combined_df[selected_features]

# Optionally, you can include the target variable 'delivery_cost' if needed
# selected_df['delivery_cost'] = combined_df['delivery_cost']
```

In [34]:

```python
import matplotlib.pyplot as plt

# Sort the selected features by importance score
sorted_selected_features = coef[coef.abs() > threshold].sort_values()

# Create a bar plot for the selected features and their importance scores
plt.figure(figsize=(10, 6))
sorted_selected_features.plot(kind="barh")
plt.title("Selected Features and Their Importance Scores")
plt.xlabel("Importance Score")
plt.ylabel("Feature")
plt.show()
```



In [35]:

```python
# Define features lists for the features
numerical_features = ["delivery_charges", 'order_total', "order_price", "distance_to_ne
```

```
categorical_features = ["istream", "is_happy_customer", "season_Spring", "season_Summer
```

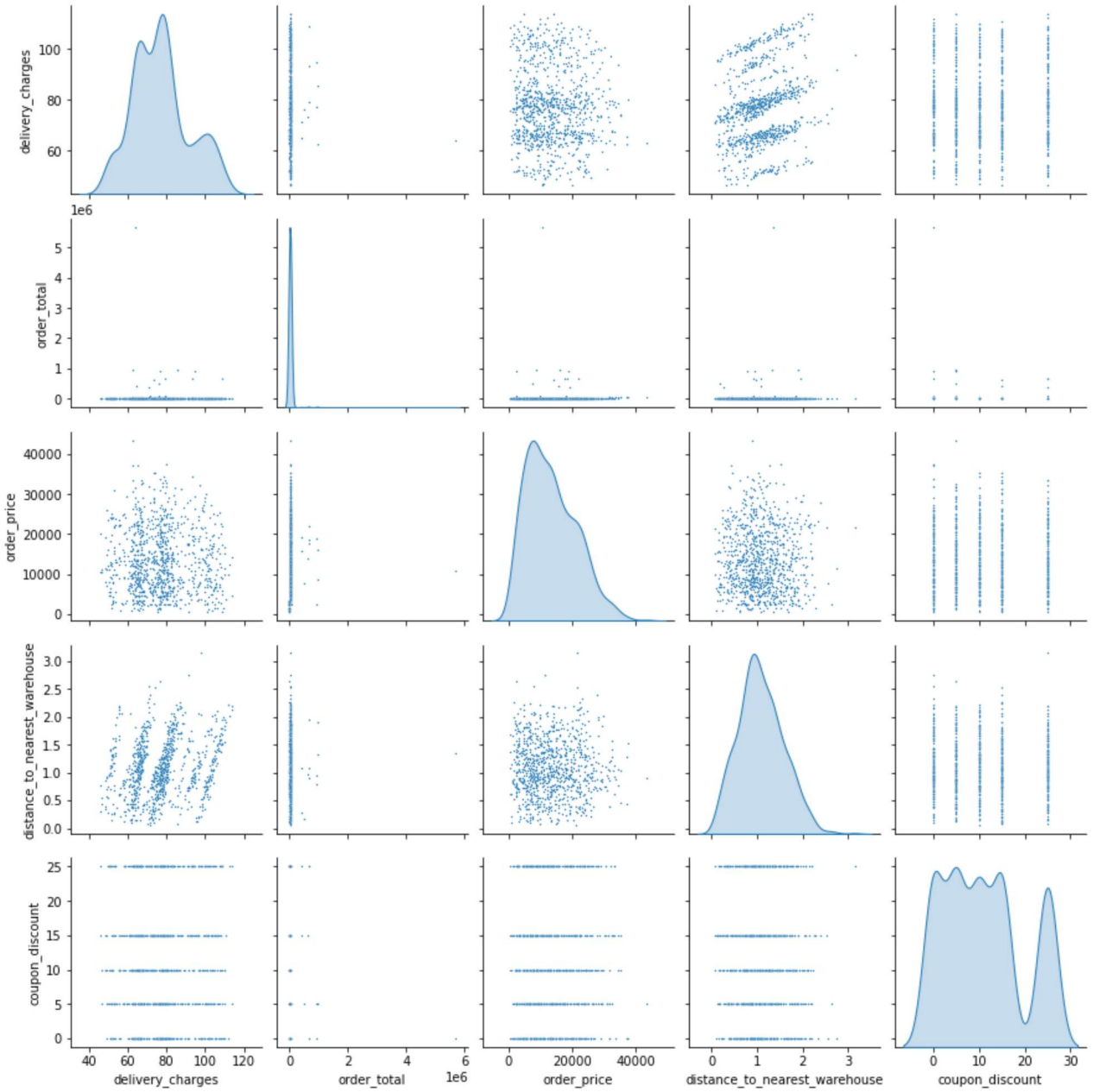
In [36]:

```
# Define a custom function for plotting histograms of the features
def describe_feature(feature_name,
                     bins=30,
                     edgecolor='k',
                     **kwargs):
    fig, ax = plt.subplots(figsize=(8,4))
    combined_df[feature_name].hist(bins=bins,
                                   edgecolor=edgecolor,
                                   ax=ax,
                                   **kwargs)
    ax.set_title(feature_name, size=15)
```

In [37]:

```
# Correlation plots between the numerical features
sns.pairplot(combined_df[numerical_features],
             plot_kws={"s": 2},
             diag_kind='kde')
```

Out[37]: <seaborn.axisgrid.PairGrid at 0x228ac89da30>



In [38]:

```
# Calculate Pearson correlation coeffiecients between the numerical values
combined_df[numerical_features].corr()
```

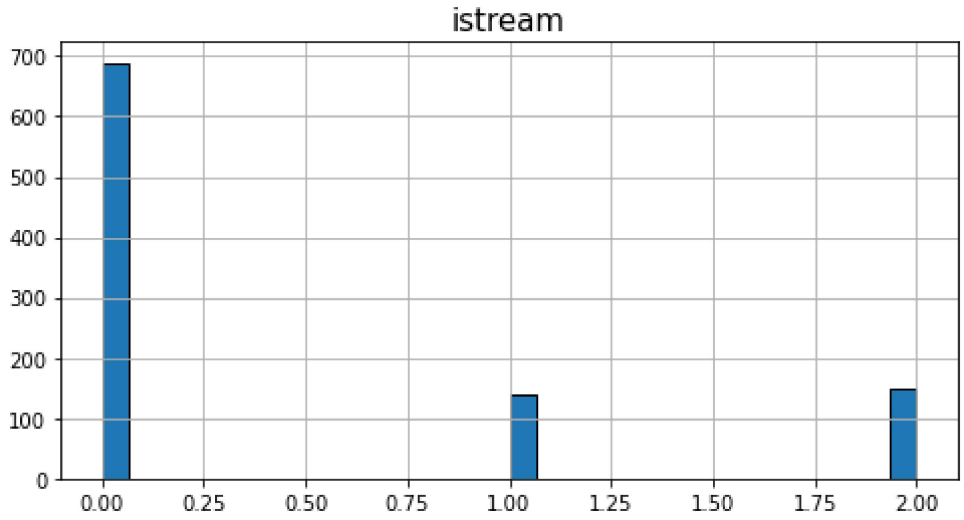
Out[38]:

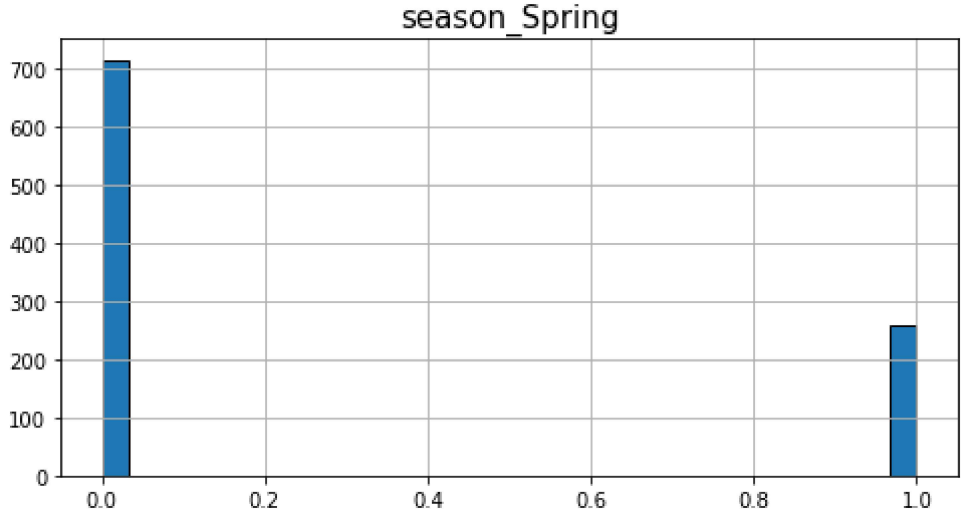
	delivery_charges	order_total	order_price	distance_to_nearest_warehouse
delivery_charges	1.000000	-0.016070	-0.033395	0.206672
order_total	-0.016070	1.000000	0.029097	0.018688
order_price	-0.033395	0.029097	1.000000	-0.017962
distance_to_nearest_warehouse	0.206672	0.018688	-0.017962	1.000000
coupon_discount	0.019091	-0.049991	-0.016032	-0.001927

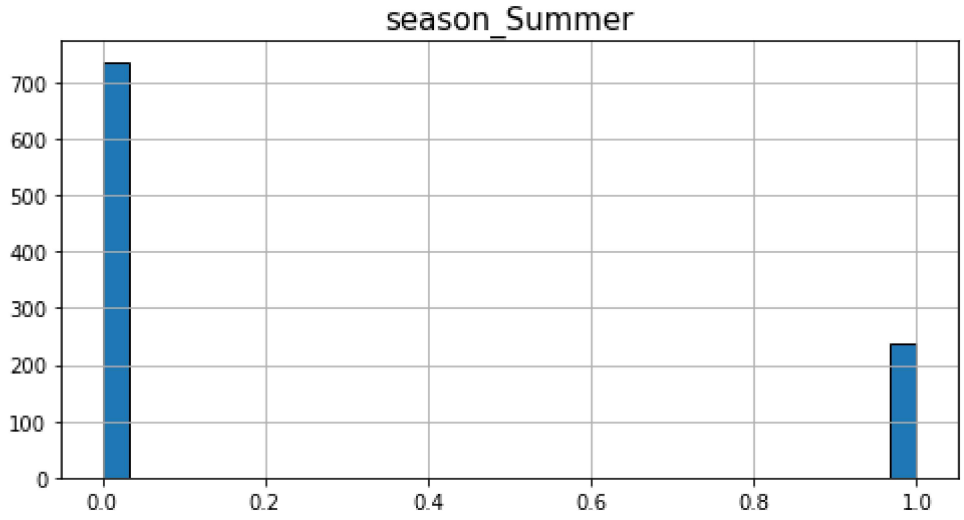
In [39]:

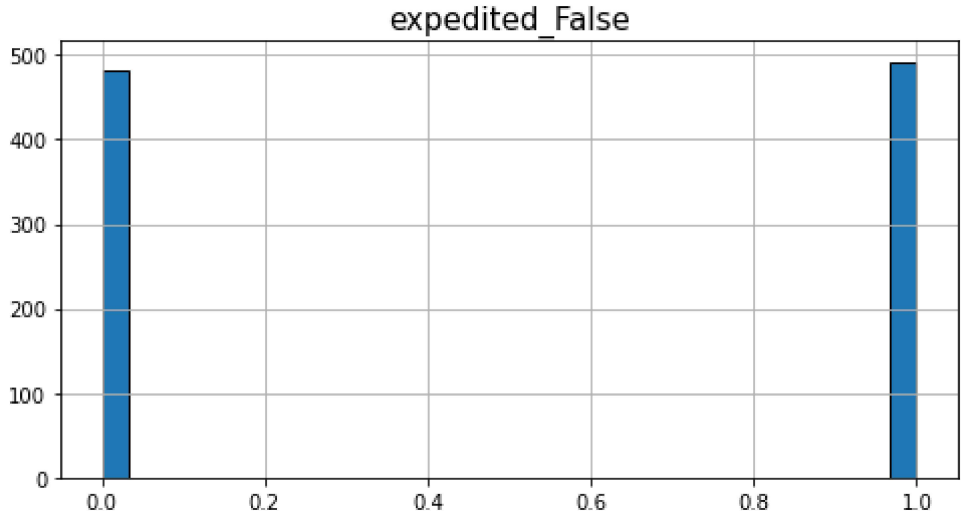
```
# Visualize the distribution of each categorical variable
```

```
for i in categorical_features:
    describe_feature(i)
```









B- Examining the Customer Locations and Warehouse Coverage

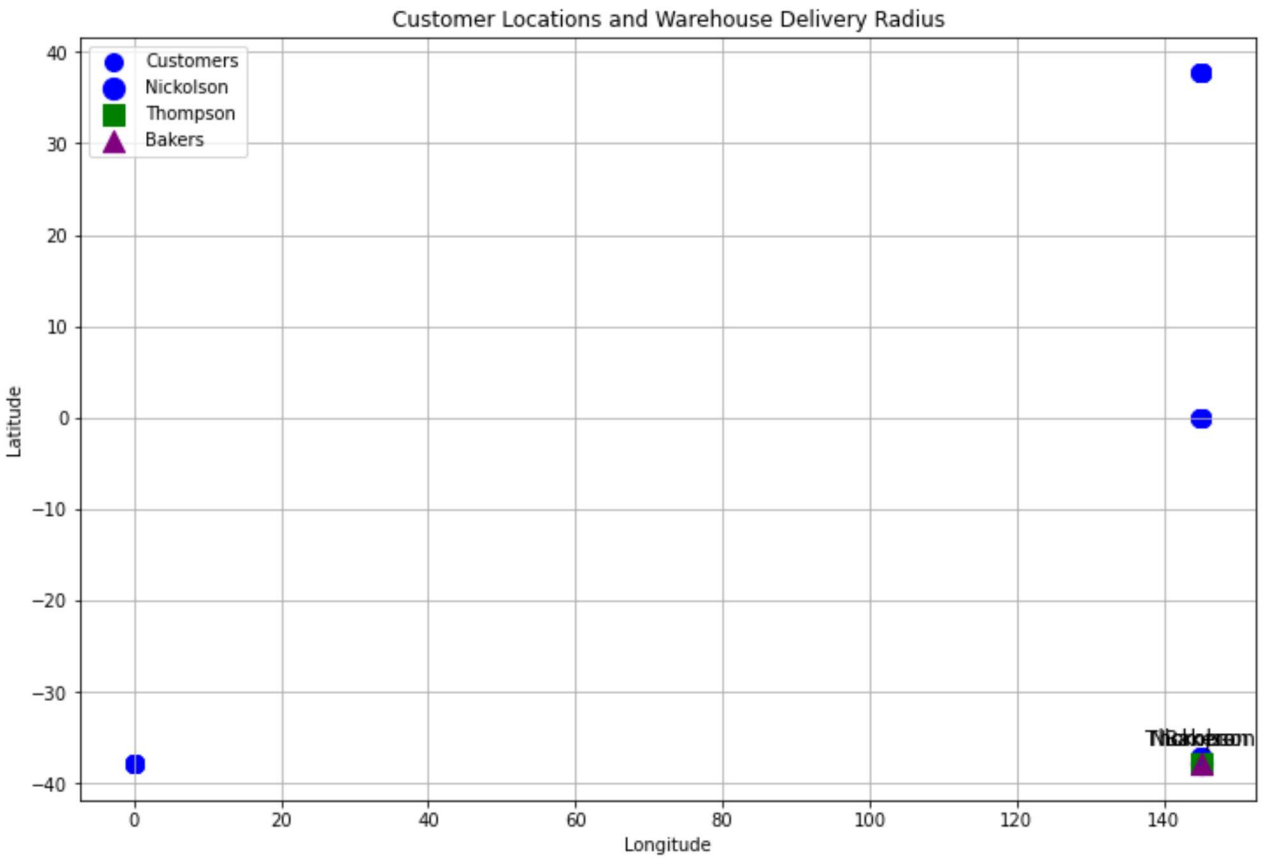
In [40]:

```
# Initialize a dictionary to store the number of customers per warehouse
customers_per_warehouse = {
    'nickolson': 0,
    'thompson': 0,
    'bakers': 0
}

# Iterate over rows in the DataFrame and count customers per warehouse
for _, row in combined_df.iterrows():
    warehouse_name = row['nearest_warehouse']
    if isinstance(warehouse_name, str):  # Check if it's a string
        warehouse_name = warehouse_name.lower()  # Convert to lowercase
        customers_per_warehouse[warehouse_name] += 1

# Display the number of customers per warehouse
for warehouse_name, num_customers in customers_per_warehouse.items():
    print(f"Warehouse: {warehouse_name.capitalize()}")  # Capitalize for display
    print(f"Number of Customers: {num_customers}\n")
```

```
Warehouse: Nickolson
Number of Customers: 353

Warehouse: Thompson
Number of Customers: 391

Warehouse: Bakers
Number of Customers: 222
```

In [41]:

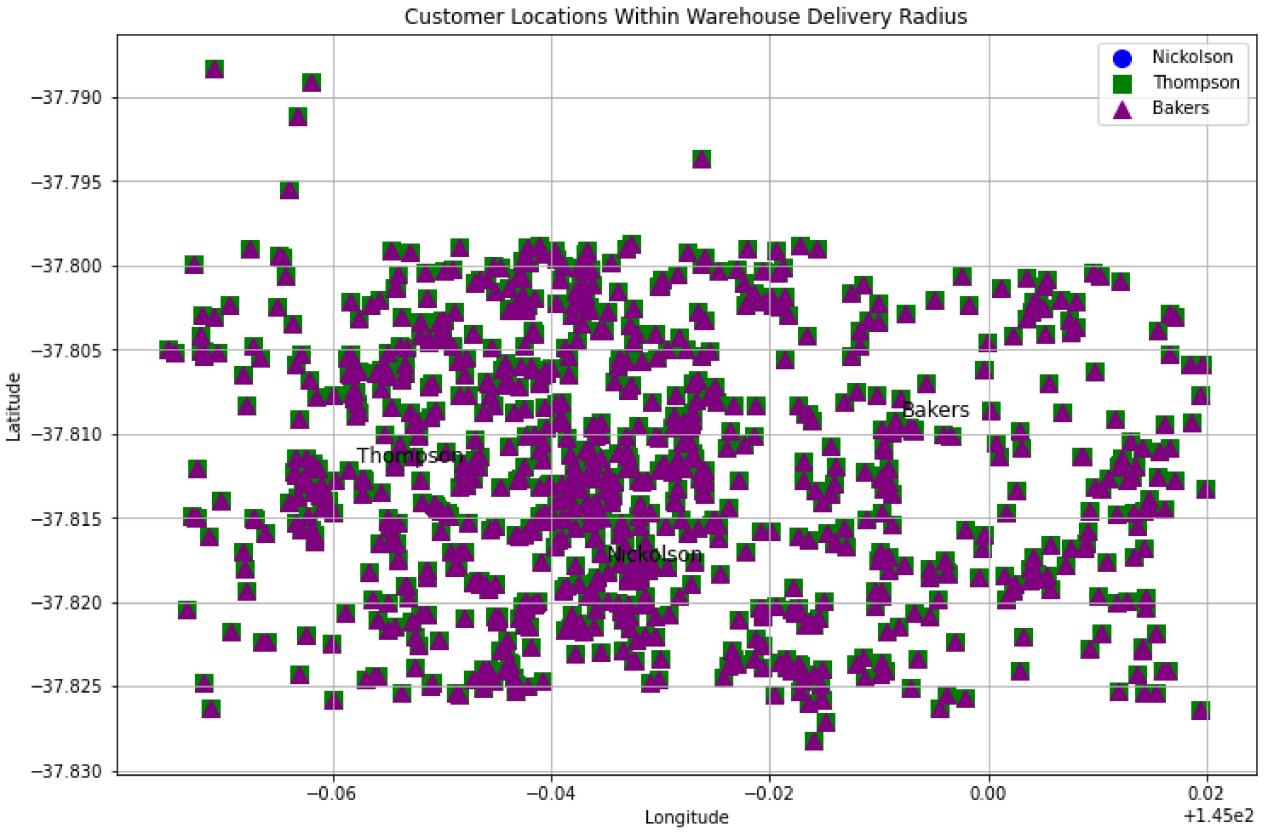
```python
# Plot customer locations
plt.figure(figsize=(12, 8))
plt.scatter(combined_df['customer_long'], combined_df['customer_lat'], c='blue', label=

# Plot warehouses with different marker styles and larger size
# The actual coordinates of the warehouses
warehouse_coordinates = [
    {'name': 'Nickolson', 'latitude': -37.818595, 'longitude': 144.969551},
    {'name': 'Thompson', 'latitude': -37.8126732, 'longitude': 144.9470689},
    {'name': 'Bakers', 'latitude': -37.8099961, 'longitude': 144.995232},
]


# Colors for the warehouses
warehouse_colors = ['blue', 'green', 'purple']  # Adjust colors as needed

# Create a buffer radius (in degrees) for the delivery radius (approximately 20 km)
buffer_radius_degrees = 0.18  # Adjust as needed

for i, warehouse in enumerate(warehouse_coordinates):
    marker_styles = ['o', 's', '^']  # Different marker styles
    plt.scatter(
        warehouse['longitude'],
        warehouse['latitude'],
        c=warehouse_colors[i],
        label=warehouse['name'],
        s=150,  # Larger marker size
        marker=marker_styles[i],  # Use different marker style
    )

    # Create a circular buffer for the delivery radius
    circle = plt.Circle((warehouse['longitude'], warehouse['latitude']), buffer_radius_
    plt.gca().add_patch(circle)

    # Add labels with warehouse names
    plt.annotate(
        warehouse['name'],
        (warehouse['longitude'], warehouse['latitude']),
        textcoords="offset points",
        xytext=(0,10),
        ha='center',
        fontsize=12,
    )

plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.title('Customer Locations and Warehouse Delivery Radius')
plt.legend()
plt.grid(True)
plt.show()
```



In [42]:

```
import matplotlib.pyplot as plt

# Plot customer locations
plt.figure(figsize=(12, 8))

# Create a buffer radius (in degrees) for the delivery radius (approximately 20 km)
buffer_radius_degrees = 0.18  # Adjust as needed

for i, warehouse in enumerate(warehouse_coordinates):
    marker_styles = ['o', 's', '^']  # Different marker styles

    # Filter customers within the delivery radius of the current warehouse
    customers_within_radius = combined_df[
        ((combined_df['customer_lat'] - warehouse['latitude']) ** 2 +
         (combined_df['customer_long'] - warehouse['longitude']) ** 2) ** 0.5 <= buffer
    ]

    # Plot customers within the radius with different colors
    plt.scatter(
        customers_within_radius['customer_long'],
        customers_within_radius['customer_lat'],
        c=warehouse_colors[i],
        label=warehouse['name'],
        s=100,  # Marker size for customers
        marker=marker_styles[i],  # Use different marker style
    )

    # Add labels with warehouse names
    plt.annotate(
        warehouse['name'],
        (warehouse['longitude'], warehouse['latitude']),
```

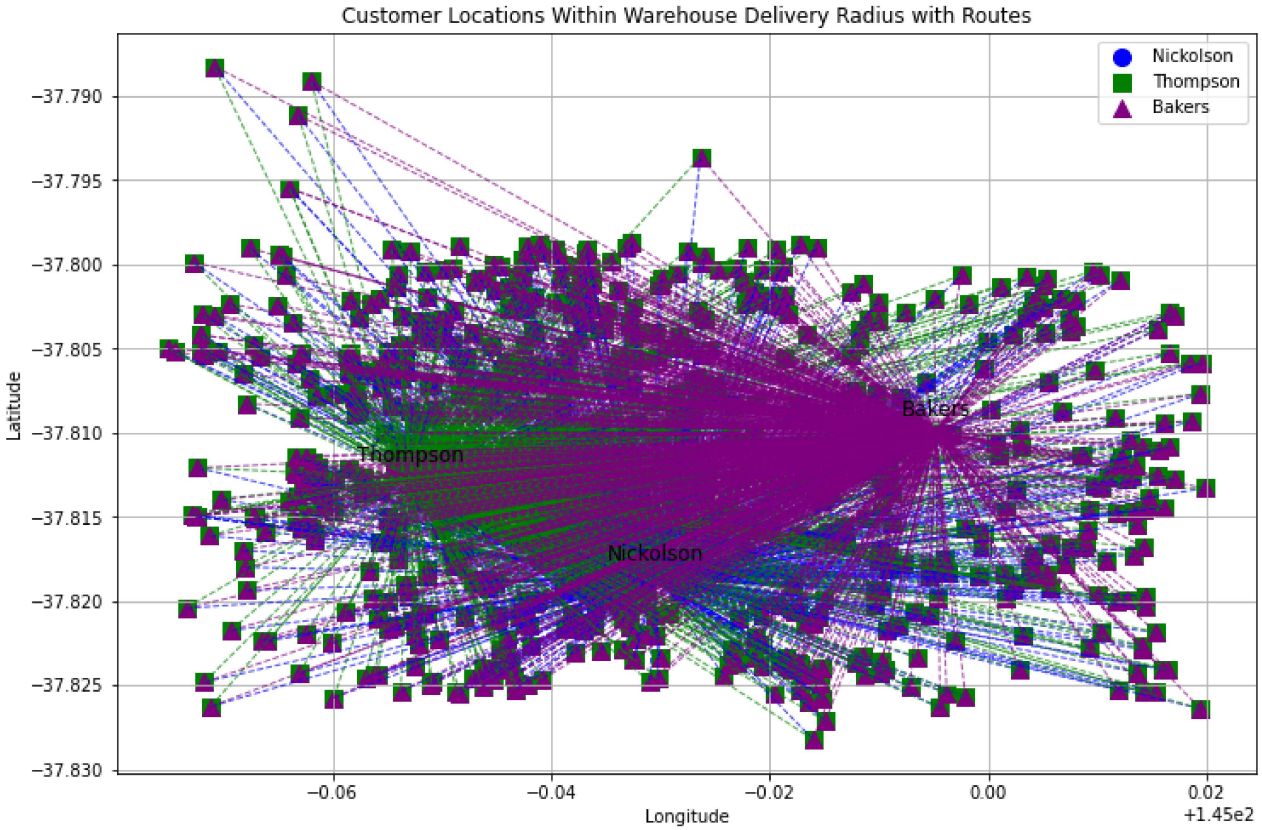
```
        textcoords="offset points",
        xytext=(0,10),
        ha='center',
        fontsize=12,
    )

plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.title('Customer Locations Within Warehouse Delivery Radius')
plt.legend()
plt.grid(True)
plt.show()
```



In [43]:

```
import matplotlib.pyplot as plt
import numpy as np

# Plot customer locations
plt.figure(figsize=(12, 8))

# Create a buffer radius (in degrees) for the delivery radius (approximately 20 km)
buffer_radius_degrees = 0.18  # Adjust as needed

for i, warehouse in enumerate(warehouse_coordinates):
    marker_styles = ['o', 's', '^']  # Different marker styles

    # Filter customers within the delivery radius of the current warehouse
    customers_within_radius = combined_df[
        ((combined_df['customer_lat'] - warehouse['latitude']) ** 2 +
         (combined_df['customer_long'] - warehouse['longitude']) ** 2) ** 0.5 <= buffer
    ]

    # Plot customers within the radius with different colors
```

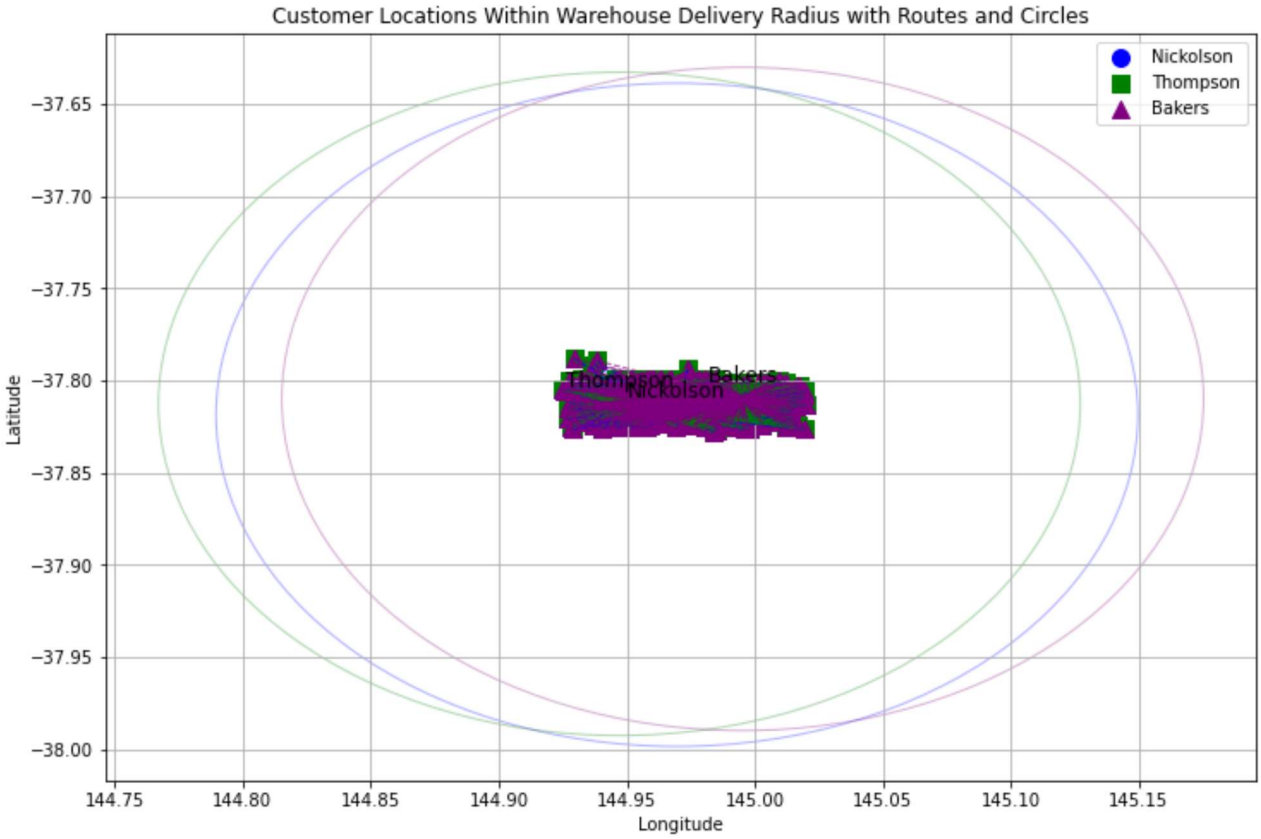
```
    plt.scatter(
        customers_within_radius['customer_long'],
        customers_within_radius['customer_lat'],
        c=warehouse_colors[i],
        label=warehouse['name'],
        s=100,  # Marker size for customers
        marker=marker_styles[i],  # Use different marker style
    )

    # Add lines connecting customers to their respective warehouses
    for _, customer in customers_within_radius.iterrows():
        plt.plot(
            [customer['customer_long'], warehouse['longitude']],
            [customer['customer_lat'], warehouse['latitude']],
            color=warehouse_colors[i],
            linestyle='--',
            linewidth=1,
            alpha=0.7,
        )

    # Add labels with warehouse names
    plt.annotate(
        warehouse['name'],
        (warehouse['longitude'], warehouse['latitude']),
        textcoords="offset points",
        xytext=(0,10),
        ha='center',
        fontsize=12,
    )

plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.title('Customer Locations Within Warehouse Delivery Radius with Routes')
plt.legend()
plt.grid(True)
plt.show()
```



In [44]:

```
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import numpy as np

# Plot customer locations
plt.figure(figsize=(12, 8))

# Create a buffer radius (in degrees) for the delivery radius (approximately 20 km)
buffer_radius_degrees = 0.18  # Adjust as needed

for i, warehouse in enumerate(warehouse_coordinates):
    marker_styles = ['o', 's', '^']  # Different marker styles

    # Filter customers within the delivery radius of the current warehouse
    customers_within_radius = combined_df[
        ((combined_df['customer_lat'] - warehouse['latitude']) ** 2 +
         (combined_df['customer_long'] - warehouse['longitude']) ** 2) ** 0.5 <= buffer
    ]

    # Plot customers within the radius with different colors
    plt.scatter(
        customers_within_radius['customer_long'],
        customers_within_radius['customer_lat'],
        c=warehouse_colors[i],
        label=warehouse['name'],
        s=100,  # Marker size for customers
        marker=marker_styles[i],  # Use different marker style
    )

    # Add lines connecting customers to their respective warehouses
    for _, customer in customers_within_radius.iterrows():
        plt.plot(
```

```
            [customer['customer_long'], warehouse['longitude']],
            [customer['customer_lat'], warehouse['latitude']],
            color=warehouse_colors[i],
            linestyle='--',
            linewidth=1,
            alpha=0.7,
        )

    # Add a circle representing the delivery radius
    circle = patches.Circle(
        (warehouse['longitude'], warehouse['latitude']),
        buffer_radius_degrees,
        fill=False,
        color=warehouse_colors[i],
        alpha=0.3,
    )
    plt.gca().add_patch(circle)

    # Add labels with warehouse names
    plt.annotate(
        warehouse['name'],
        (warehouse['longitude'], warehouse['latitude']),
        textcoords="offset points",
        xytext=(0,10),
        ha='center',
        fontsize=12,
    )

plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.title('Customer Locations Within Warehouse Delivery Radius with Routes and Circles'
plt.legend()
plt.grid(True)
plt.show()
```


Customer Locations Within Warehouse Delivery Radius with Routes and Circles
Nickolson
Thompson
Bakers
Latitude
Longitude
−37.65
−37.70
−37.75
−37.80
−37.85
−37.90
−37.95
−38.00
144.75
144.80
144.85
144.90
144.95
145.00
145.05
145.10
145.15
Thompson
Bakers
Nickolson