Kite Packaging

Courier Cost Saving Analysis

Analytical Avengers

12th June 2023

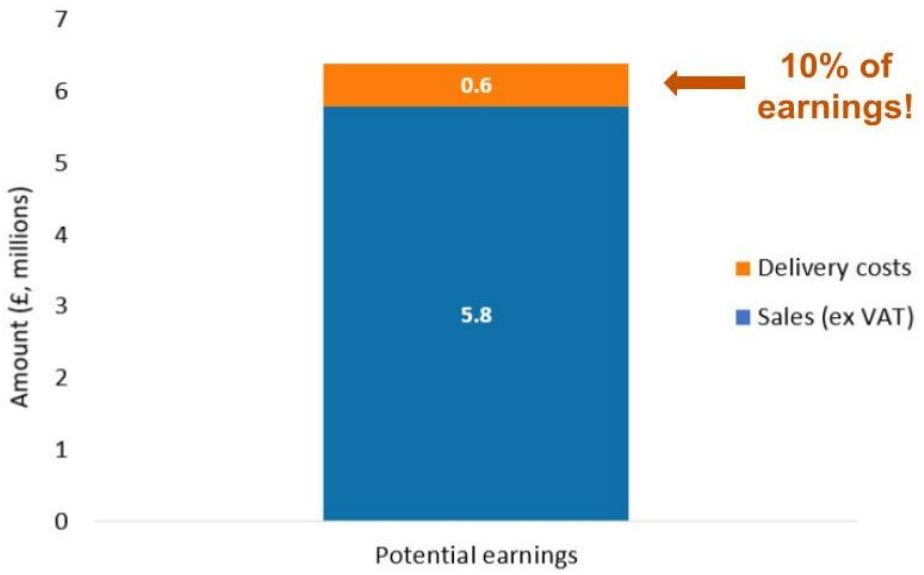
Table of Contents

Table of Figures

1. Business Context and Problem

Kite Packaging currently relies on third-party couriers for its logistics operations, leading to a considerable proportion of earnings spent on delivery costs.

Figure 1: Kite spends a considerable proportion of earnings on delivery costs.



Source: Kite Packaging; based on raw data from March 27, 2023, to April 22, 2023, for its Coventry distribution centre.

There are additional non-financial benefits to bringing deliveries in-house that should be considered, including improved company brand and reputation, control over delivery times and quality, and increased customer satisfaction.

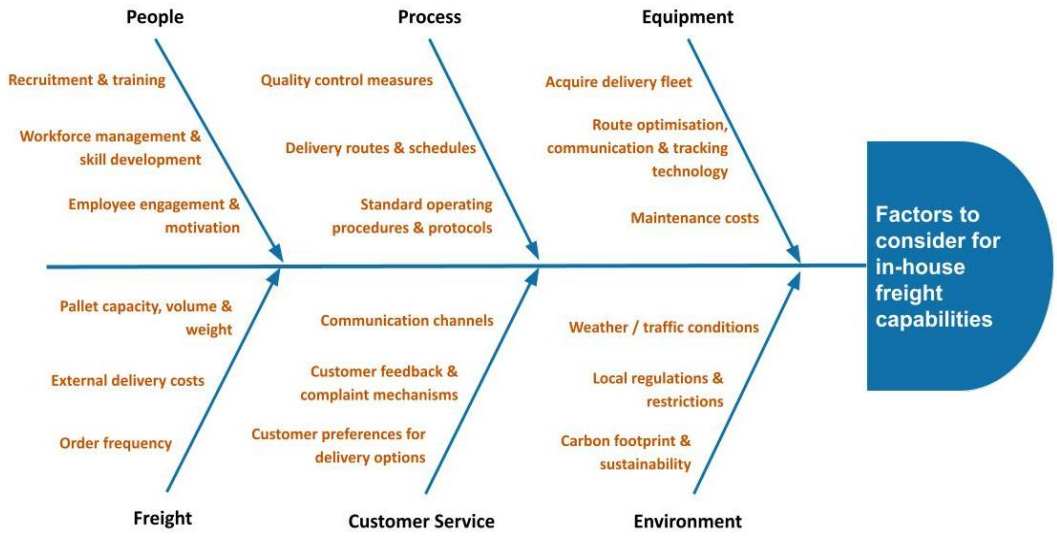
Figure 2: Benefits of bringing deliveries in-house.

Source: Analytical Avengers.

This analysis will investigate the feasibility of expanding in-house logistics capabilities to reduce freight costs by 20% within a 50-mile radius around its Coventry distribution centre. Considering various factors for building in-house freight capabilities, the following questions were identified to support the investigation:

- How are customers clustered around the distribution centre?
- What is the largest cluster by orders and delivery costs?
- Who are the most expensive couriers, and which have the highest delivery costs?
- What is the most cost-effective and low-risk approach to bring deliveries in-house?

Figure 3: Factors to consider for building in-house freight capabilities.



Source: Analytical Avengers.

2. Project Development Process

2.1. Data Cleaning and Manipulation Approach

Three datasets, with historic data from a three-week period (27-Mar-23 to 22-Apr-23), were provided by Kite:

- **'Customers'**: customer location details.
- **'Orders'**: cost of orders, order and dispatch dates, order status, courier information.
- **'Order Details'**: product details of orders.

An additional data source was incorporated by scraping Kite Trustpilot reviews, to gain insights into customer sentiment and identify delivery-related issues.

2.1.1. Python Libraries and Functions

Pandas, NumPy, Matplotlib, Seaborn, Statsmodels, and Scikit-learn were used for data cleaning and manipulation, visualisation, statistical modelling, model selection, and clustering. Natural language processing (NLP) functions were employed for sentiment analysis, and enabled efficient handling, analysis, and visualisation of Trustpilot data.

2.1.2. Data Quality Checks

Several data checks were performed on the datasets:

- Data types were checked for consistency, e.g., date formats were standardised.
- Columns were renamed for clarity.
- Null values were addressed[1].
- In Orders, rows with 'Deleted' order status were removed.
- SubTotal and GrandTotal in Orders were cross-referenced to ensure data integrity.
- Weekends and bank holidays were removed as a business month was used.
- Datasets were checked for duplicates, of which none were found[2].

[1] Missing and null values were addressed using appropriate methods, including imputation and deletion, based on the specific context in each dataset. This considered factors such as data availability, impact on analysis, and the preservation of data integrity. While a full description of missing value handling is not provided for brevity, appropriate data cleaning techniques were applied to minimise the impact of missing and null values on analysis.

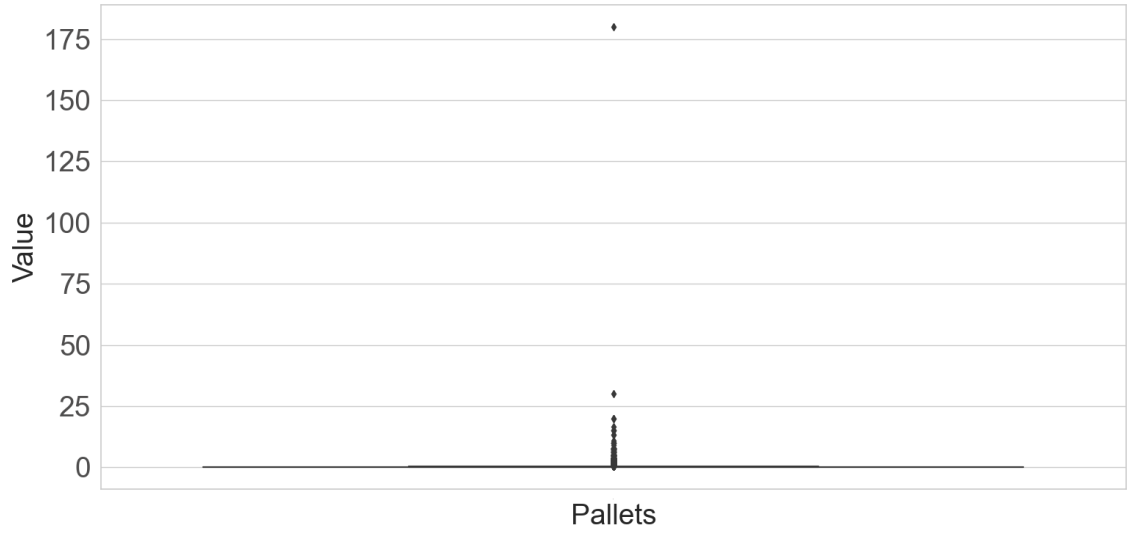
[2] There were 10 duplicate rows in the Orders Details dataset. To check whether these were genuinely part of an order, the sum of each order product row was cross checked against the 'Orders' dataset. This revealed there were no 'true duplicates' in the data.

2.1.3. Outliers

Boxplots were used to visualise outliers:

- In Orders, outliers at the upper end for order value/delivery costs were retained as they represent valid data points of the most valuable customers.
- In Order Details, one significant outlier for number of pallets was removed, as it was unrepresentative of typical orders and its inclusion would have skewed insights.

Figure 4: Outlier for pallets in Order Details dataset.



Source: Analytical Avengers.

2.1.4. Postcode Data Enrichment

Coordinates were batch-generated for each postcode in Customers by extracting outcodes and enriching data using a CSV file of coordinate values[3].

- Three postcodes remained unresolved and were removed as they represented a minimal proportion of the total.

[3] The CSV file of latitude and longitude for each postcode was obtained through this link:

UK Postcodes – Distance Calculator - 101 Computing

2.2. Visualisation Approach

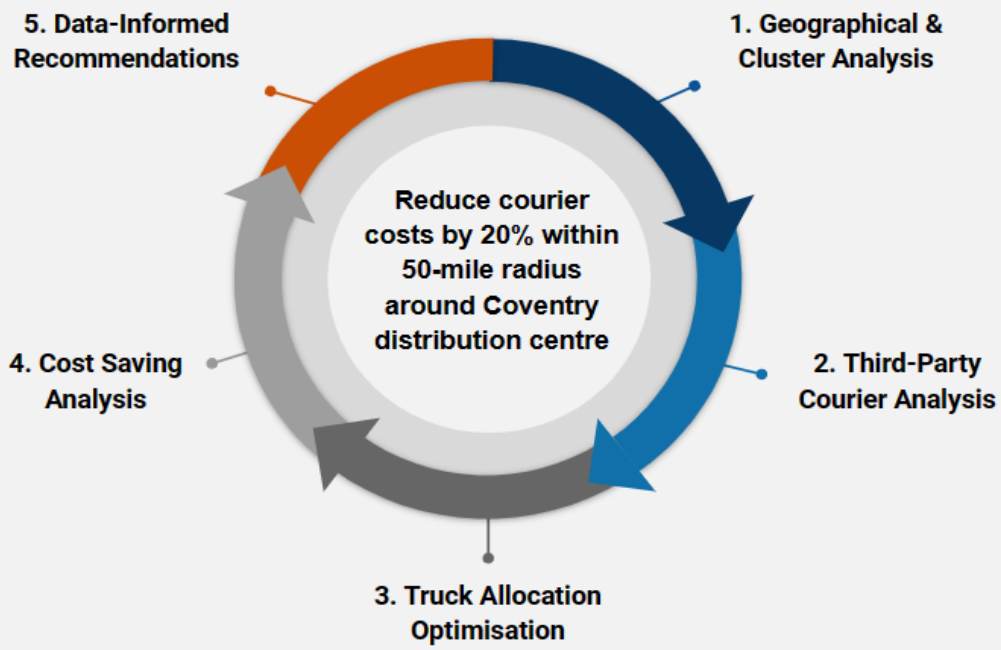
Visualisations were standardised for all deliverables, ensuring consistency and accessibility throughout.

- **Python, Tableau, and Excel:** Used to create visualisations and enhance data exploration and presentation of data.
- **Visualisation type:** Bar charts enabled effective analysis and comparison of freight, such as delivery costs and orders by courier and cluster. Symbol maps were used to visualise the geographical clustering of customers.
- **Colour palette**: All deliverables used Tableau's 'Colorblind' palette for consistency. This palette, with prescribed contrast ratios, accommodates colour vision deficiencies.
- **Size:** Visuals were optimised with appropriate size for easy comprehension, ensuring legible text, and sensible sizing of data points.
- **Accessibility:** Visualisations included readable font sizes, contrasting colours, and annotations for context. All axes, measures, and dimensions have meaningful labels.

2.3. Data Analysis Approach

The data analysis approach involved visualising customer distribution and creating clusters, investigating couriers by cost, creating a truck allocation model, and simulating cost savings.

Figure 5: Approach to cost saving analysis.

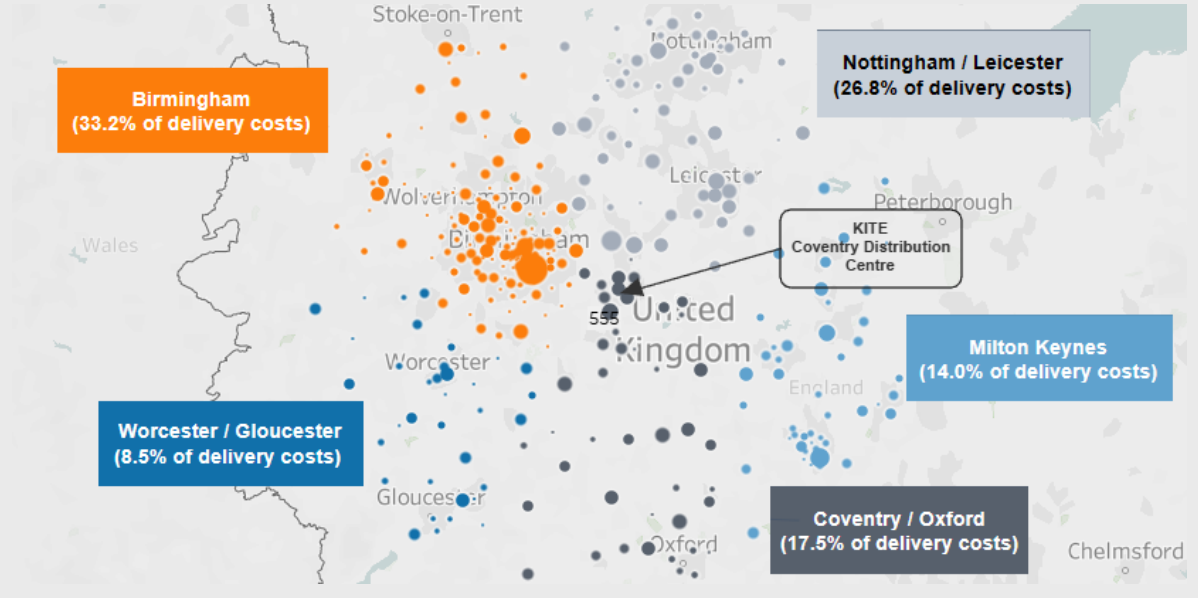


Source: Analytical Avengers.

2.3.1. Geographical Analysis

Given Kite's business objective, postcodes were filtered to a 50-mile radius around the Coventry distribution centre. Five customer clusters, located around major cities or towns, were created to identify potential areas for a pilot study. The Birmingham cluster had the highest proportion of delivery costs (33%), followed by Nottingham/Leicester (27%).

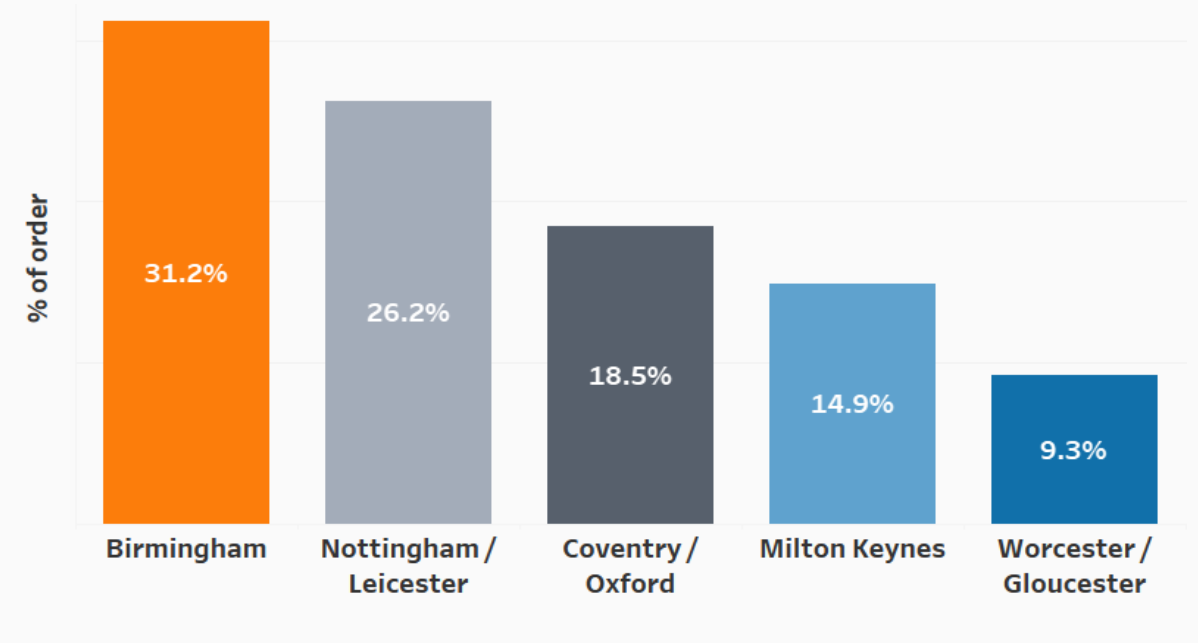
Figure 6: Geographical distribution of customers revealed five clusters.



Source: Kite Packaging, Analytical Avengers; based on delivery costs from 27-Mar-23 to 22-Apr-23.
Note: Size of dots represents sum of delivery costs

This trend was similar for the breakdown of total orders across clusters, with Birmingham having the highest proportion (31%), followed by Nottingham/Leicester (26%).

Figure 7: Percentage of orders by cluster.

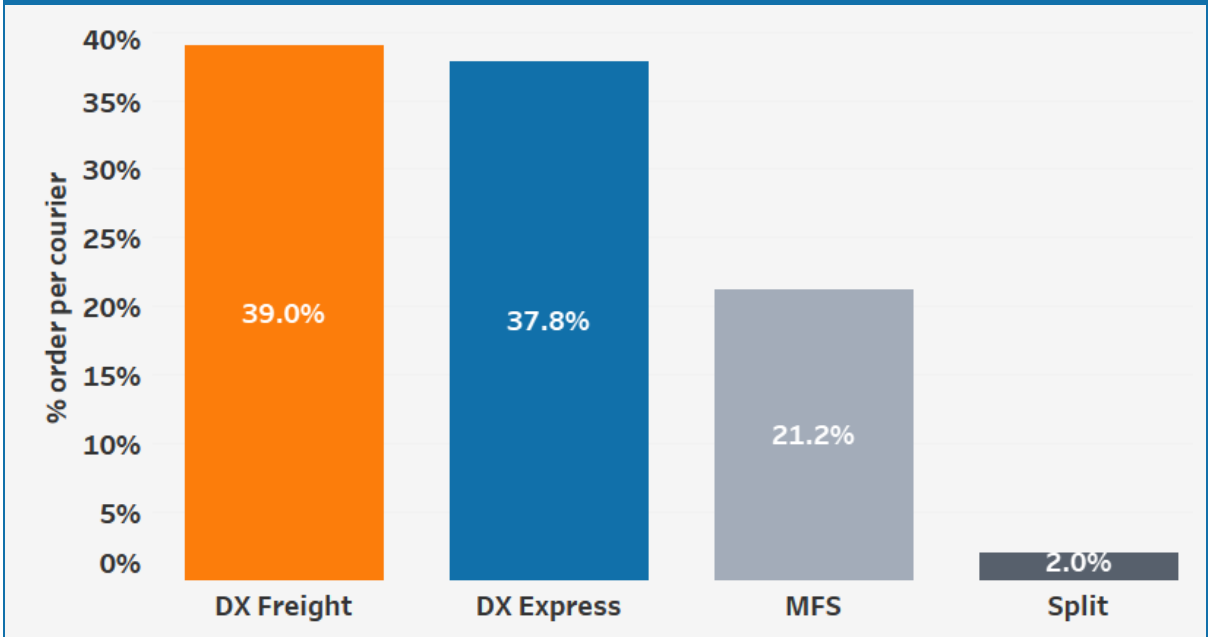


Source: Kite Packaging, Analytical Avengers; based on orders from 27-Mar-23 to 22-Apr-23.

2.3.2. Courier Analysis

Initial analysis revealed that DX Freight and DX Express had the highest number of orders, at 39% and 38% respectively.

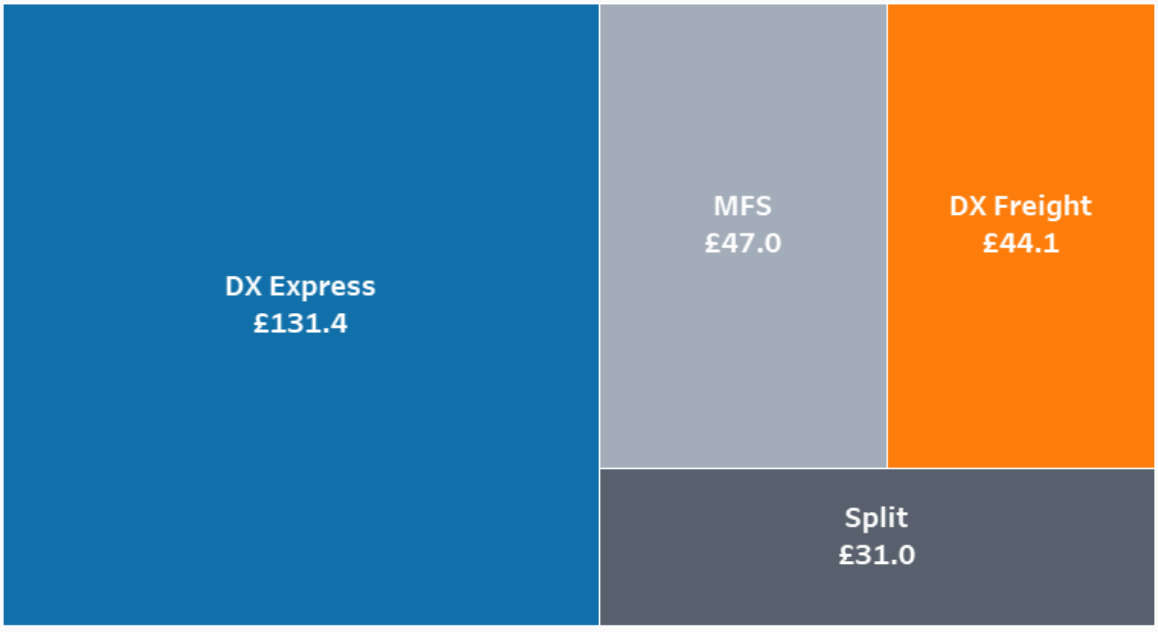
Figure 8: DX Freight and DX Express have the highest number of orders.



Source: Kite Packaging, Analytical Avengers; based on orders from 27-Mar-23 to 22-Apr-23.

Focusing on cost per pallet, DX Express was the most expensive at over £130 per pallet, likely due to its high-frequency, small parcel delivery model.

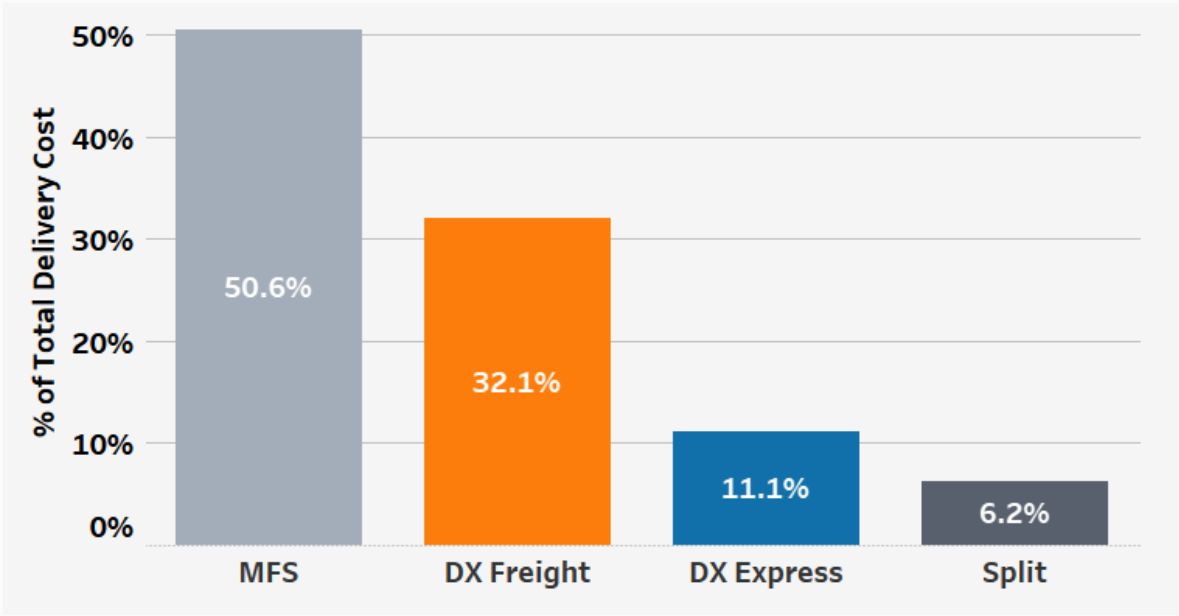
Figure 9: DX Express was found to have the highest cost per pallet.



Source: Kite Packaging, Analytical Avengers; based on delivery costs from 27-Mar-23 to 22-Apr-23.
Note: DX Express does not deliver pallets so cost per pallet was estimated using volume.

However, further analysis revealed that MFS and DX Freight represent well over 80% of total delivery costs, whereas DX Express represented just 11%.

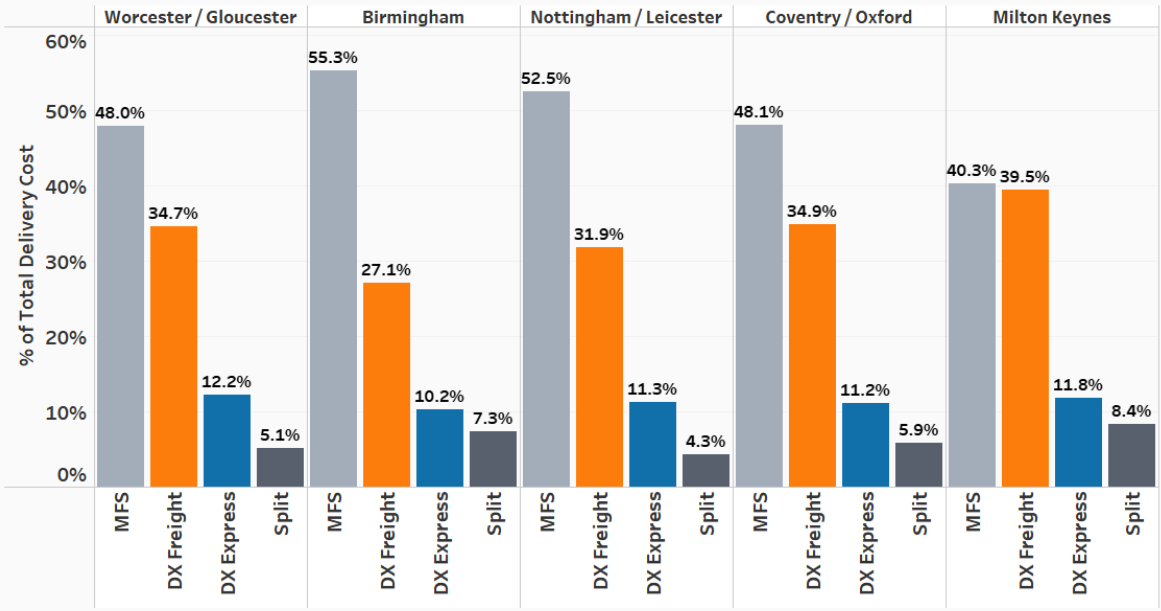
Figure 10: MFS and DX Freight have the highest proportion of delivery costs.



Source: Kite Packaging, Analytical Avengers; based on delivery costs from 27-Mar-23 to 22-Apr-23.

A similar distribution of delivery costs by couriers was seen across the clusters. Considering these findings, a decision was made to retain DX Express deliveries and investigate replacing other couriers' deliveries, as this offered greater cost-cutting opportunities.

Figure 11: Distribution of delivery cost per courier similar across clusters.

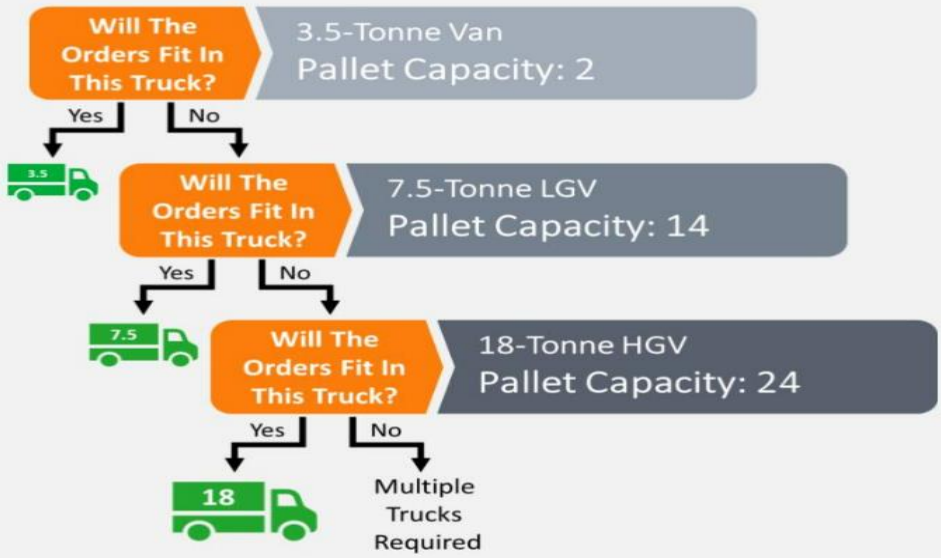


Source: Kite Packaging, Analytical Avengers; based on delivery costs from 27-Mar-23 to 22-Apr-23.

2.3.3. Truck Allocation Optimisation

A function was created to identify the number of trucks needed based on specific pallet quantities to optimise truck allocation for each cluster. This function takes a given number of pallets and selects the smallest truck these can fit into, preferring one large truck over multiple smaller trucks for cost-effectiveness.

Figure 12: Truck allocation model.

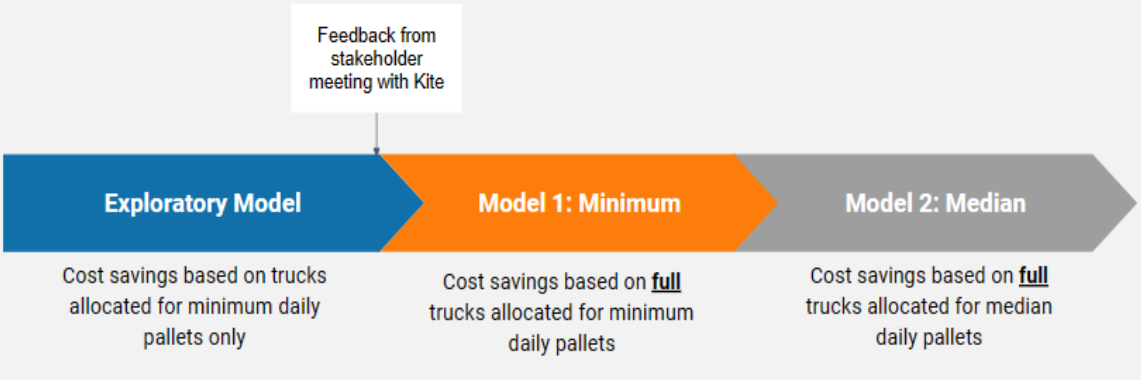


Source: Analytical Avengers.

2.3.4. Cost Saving Analysis

The exploratory model determined savings based on delivering the minimum daily orders for a month, minimising financial risk by only committing to the number of pallets that Kite was almost certainly guaranteed to have to deliver. However, trucks were not filled and too many orders were allocated to couriers, missing opportunities to save on costs. Therefore, the model was enhanced to fill in-house trucks where possible (Model 1), simulating a more realistic picture of how in-house deliveries would be conducted. Analysis was repeated using median daily pallets (Model 2), to see if committing to additional fleet costs would be offset by greater cost savings.

Figure 13: Original model was optimised to create more cost savings.



Source: Analytical Avengers.

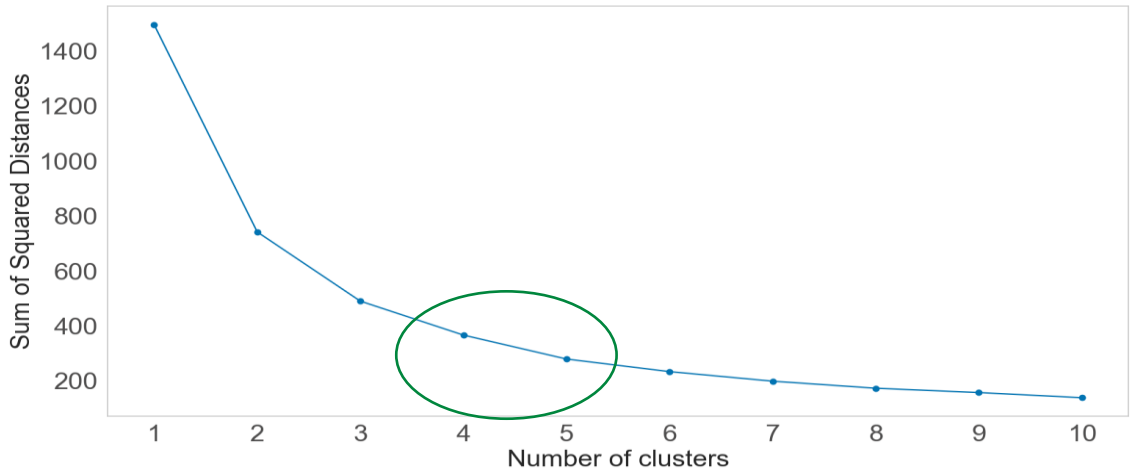
3. Technical Overview of the Code

3.1. Geographic Clustering

Customers' coordinates were geographically mapped in Python and the mapping was then visualised in both Python and Tableau. The mapping revealed:

1. Customers were dispersed outwards from midpoints around major towns and cities.
2. Data point density, representing the high concentration of customers and associated delivery costs, indicated the potential to identify a location to run a pilot scheme.

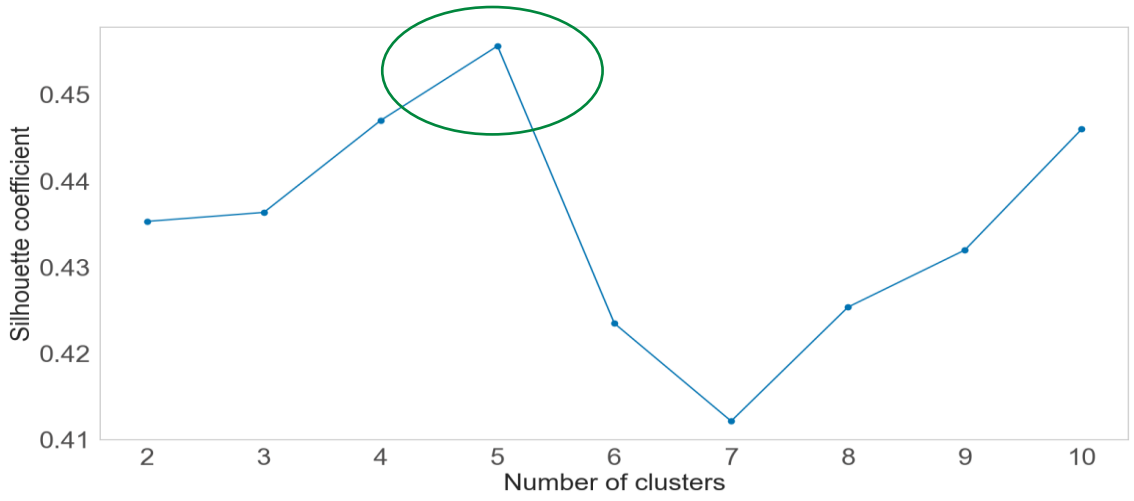
Figure 14: The Elbow method suggests 4-5 clusters.



Source: Analytical Avengers.

To determine the optimal number of location groups (clusters), the Elbow and Silhouette methods were employed. The Elbow method indicated four or five would be the optimal number of clusters, while the Silhouette method suggested five.

Figure 15: The Silhouette method suggests 5 clusters.



Source: Analytical Avengers.

Employing a 'K-means' clustering algorithm to separate the data observations into groups suggested that five clusters would be more suitable, with each cluster covering a similarly sized geographic area[4].

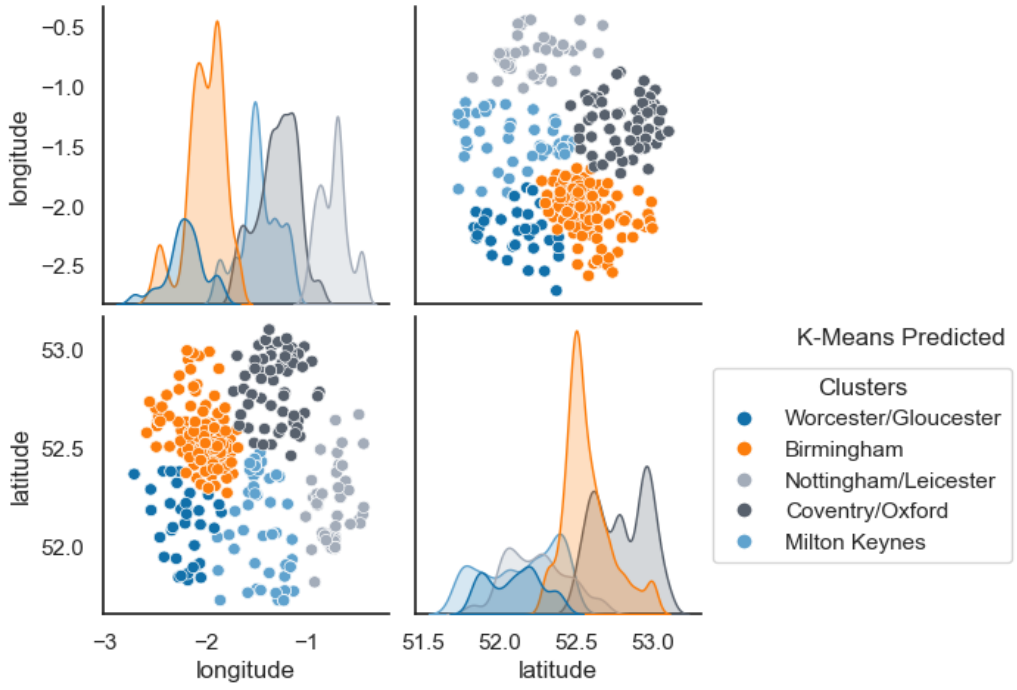
Figure 16: Code for 'K-means' clustering.

```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

# Use KMeans with five clusters.
kmeans2 = KMeans(n_clusters=5,
                 max_iter=15000,
                 init='k-means++',
                 random_state=42).fit(b)

clusters5 = kmeans2.labels_
b['K-Means Predicted'] = clusters5

# Plot the predicted clusters with modified hue parameter and disable the default legend.
sns.pairplot(b,
             hue='K-Means Predicted',
             diag_kind='kde',
             plot_kws={'legend': False})
```

Source: Analytical Avengers.

Tableau was used to identify the five geographical locations corresponding to each cluster: Worcester/Gloucester, Birmingham, Nottingham/Leicester, Coventry/Oxford, Milton Keynes.

Figure 17: K-means model with five clusters.



Source: Kite, Analytical Avengers.

Geographical clustering provided valuable insights for decision-making for integrating in-house deliveries. By grouping customers based on their geographic proximity, clustering techniques facilitate the optimisation of freight operations by identifying cost-saving opportunities, enhancing delivery routes, and improving efficiency.

[4] We tested four and five clusters in the K-means clustering and concluded that five clusters best represented our interpretation of the data, as these represented five distinct areas of high customer concentration.

3.2. Freight Analysis and Truck Allocation

Analysis was based on the core objective of partially replacing couriers, to avoid commitment to excess capacity. We started by exploring the number and type of vehicles required to guarantee delivery of the minimum number of daily pallets in a month. To this end, a truck allocation function in Python was created. Our first function had limitations in that it could only allocate a maximum of one truck to a cluster. This was refined to be able to allocate a maximum of two trucks to a cluster, as shown below.

Figure 18: Code for truck allocation function.

```python
# Create a function to allocate trucks to each cluster based on the number of pallets needed to be delivered.

def truck_allocation(pallets):

    # Define the maximum pallet capacity of each type of truck and truck combination.
    pallet_capacity_3_5_tonne_van = 2
    pallet_capacity_7_5_tonne_LGV = 14
    pallet_capacity_18_tonne_HGV = 24
    pallet_capacity_18_3_5_tonne_trucks = 26
    pallet_capacity_7_5_7_5_tonne_trucks = 28
    pallet_capacity_18_7_5_tonne_trucks = 38
    pallet_capacity_18_18_tonne_trucks = 48

    # Create a DataFrame to store the truck type and number of trucks needed for each cluster.
    df_trucks_needed = pd.DataFrame(index = pallets.index)
    # Add columns to store the information needed.
    df_trucks_needed['Pallets'] = pallets
    df_trucks_needed['3.5-tonne Van (2)'] = 0
    df_trucks_needed['7.5-tonne LGV (14)'] = 0
    df_trucks_needed['18-tonne HGV (24)'] = 0

    # Use a for loop to determine the truck type needed for the number of pallets in each cluster.
    for cluster in df_trucks_needed.index:

        # Identify the pallets needed for that cluster.
        pallets_needed = df_trucks_needed.loc[cluster, 'Pallets']

        # Check if the pallets needed can fit into a 3.5-tonne van.
        if pallets_needed <= pallet_capacity_3_5_tonne_van:
            df_trucks_needed.loc[cluster, '3.5-tonne Van (2)'] = 1

        # If not, can they fit into a 7.5-tonne LGV.
        elif pallet_capacity_3_5_tonne_van < pallets_needed <= pallet_capacity_7_5_tonne_LGV:
            df_trucks_needed.loc[cluster, '7.5-tonne LGV (14)'] = 1

        # If not, can they fit into an 18-tonne HGV.
        elif pallet_capacity_7_5_tonne_LGV < pallets_needed <= pallet_capacity_18_tonne_HGV:
            df_trucks_needed.loc[cluster, '18-tonne HGV (24)'] = 1

        # If not, can they fit into an 18-tonne HGV and a 3.5-tonne van.
        elif pallet_capacity_18_tonne_HGV < pallets_needed <= pallet_capacity_18_3_5_tonne_trucks:
            df_trucks_needed.loc[cluster, '18-tonne HGV (24)'] = 1
            df_trucks_needed.loc[cluster, '3.5-tonne Van (2)'] = 1

        # If not, can they fit into two 7.5-tonne LGVs.
        elif pallet_capacity_18_3_5_tonne_trucks < pallets_needed <= pallet_capacity_7_5_7_5_tonne_trucks:
            df_trucks_needed.loc[cluster, '7.5-tonne LGV (14)'] = 2

        # If not, can they fit into an 18-tonne HGV and a 7.5-tonne LGV.
        elif pallet_capacity_7_5_7_5_tonne_trucks < pallets_needed <= pallet_capacity_18_7_5_tonne_trucks:
            df_trucks_needed.loc[cluster, '18-tonne HGV (24)'] = 1
            df_trucks_needed.loc[cluster, '7.5-tonne LGV (14)'] = 1

        # If not, can they fit into two 18-tonne HGVs.
        elif pallet_capacity_18_7_5_tonne_trucks < pallets_needed <= pallet_capacity_18_18_tonne_trucks:
            df_trucks_needed.loc[cluster, '18-tonne HGV (24)'] = 2

    # Add a row to the DataFrame that sums the number of trucks needed for each truck type.
    df_trucks_needed.loc['Total'] = df_trucks_needed.sum()

    # Return the DataFrame with the truck allocation for each cluster.
    return df_trucks_needed
```

Source: Analytical Avengers.

The function used the maximum capacity of each of the three truck types[5] as inputs. With the knowledge that one big truck is cheaper than two smaller trucks, the function was designed to determine the most cost-efficient truck or combination of trucks needed to deliver a minimum daily number of pallets for each cluster[6].

Figure 19: Truck allocation by cluster for minimum daily pallets.

Cluster Label	Pallets	3.5-tonne Van (2)	7.5-tonne LGV (14)	18-tonne HGV (24)
Worcester / Gloucester	4.1	0	1	0
Birmingham	19.2	0	0	1
Nottingham / Leicester	18.1	0	0	1
Coventry / Oxford	9.4	0	1	0
Milton Keynes	9.0	0	1	0
Total	59.8	0	3	2

Source: Analytical Avengers.

At a later stage of the project, it was decided to widen the scope of our analysis, and therefore the truck allocation function was used to determine the trucks needed to deliver median daily pallets. The median was selected as a measure of average instead of the mean due to the skewed distribution and presence of outliers in the number of daily pallets.

Figure 20: Truck allocation by cluster for median daily pallets.

Cluster Label	Pallets	3.5-tonne Van (2)	7.5-tonne LGV (14)	18-tonne HGV (24)
Worcester / Gloucester	8.5	0	1	0
Birmingham	36.9	0	1	1
Nottingham / Leicester	27.4	0	2	0
Coventry / Oxford	20.9	0	0	1
Milton Keynes	16.2	0	0	1
Total	110.0	0	4	3

Source: Analytical Avengers.

[5] Data on truck types and associated costs supplied by Kite.

[6] 'Pallets' were decided as the primary constraint. The 'Pallets' variable is highly correlated with other constraints (weight and volume).

3.3.Financial Analysis

Following truck allocation, a function was created in Python to calculate potential cost-savings of delivering minimum daily pallets. This calculated the cost of the allocated trucks, using running and standing costs provided by Kite. Next, it calculated the cost of outsourcing remaining deliveries to couriers. This simulated cost was then compared to current incurred delivery costs to calculate cost savings for each cluster. While our exploratory cost-saving model worked, it had limitations as it did not fill the truck allocated to deliver the minimum daily pallets, thereby missing potential cost savings. We decided to refine our function to use the capacity of trucks allocated, as seen below.

Figure 21: Code for cost savings function - Part 1

```
# Function to calculate the cost savings of a particular truck allocation model

def cost_savings(model):

    # Looking at the number of pallets delivered each day for each cluster
    pallets_day = kite_nonDX.pivot_table(index='Cluster Label',
                                         columns='DateDespatched',
                                         values='Pallets',
                                         aggfunc='sum').transpose()

    # Using a for loop to work out the number of pallets that can be delivered each day in each cluster
    # Using the model to define the truck capacity in each cluster
    for cluster in range(0,5):
        capacity = ((model.loc[cluster, '3.5-tonne Van (2)']*2) +
                    (model.loc[cluster, '7.5-tonne LGV (14)']*14) +
                    (model.loc[cluster, '18-tonne HGV (24)']*24))
        pallets_day.iloc[:, cluster] = (pallets_day.iloc[:, cluster].apply(
                                        lambda x : capacity if x > capacity else x))

    # Summing the pallets that can be delivered by trucks in each cluster
    Pallets_truck = pallets_day.sum()
    # Scaling this up to a month
    Pallets_truck_month = Pallets_truck*22/18
    # Storing this information in a DataFrame
    Pallets_truck_month_df = Pallets_truck_month.to_frame()
    # Changing column names to be more appropriate
    Pallets_truck_month_df = Pallets_truck_month_df.rename(
                               columns={0: 'Pallets delivered by in-house trucks'})


    # Now working out the cost of the trucks in each cluster:

    # Defining the running and standing costs for each truck type
    standing_costs_3_5_tonne_van = 600
    standing_costs_7_5_tonne_LGV = 1250
    standing_costs_18_tonne_HGV = 1800
    running_costs_3_5_tonne_van = 2800
    running_costs_7_5_tonne_LGV = 3200
    running_costs_18_tonne_HGV = 3600

    # Creating a copy of the model
    truck_costs = model

    # Adding columns to store the running and standing costs for the trucks in each cluster
    truck_costs['3.5_tonne_standing_costs'] = (truck_costs['3.5-tonne Van (2)']*
                                               standing_costs_3_5_tonne_van)
    truck_costs['7.5_tonne_standing_costs'] = (truck_costs['7.5-tonne LGV (14)']*
                                               standing_costs_7_5_tonne_LGV)
    truck_costs['18_tonne_standing_costs'] = (truck_costs['18-tonne HGV (24)']*
                                              standing_costs_18_tonne_HGV)
    truck_costs['3.5_tonne_running_costs'] = (truck_costs['3.5-tonne Van (2)']*
                                              running_costs_3_5_tonne_van)
    truck_costs['7.5_tonne_running_costs'] = (truck_costs['7.5-tonne LGV (14)']*
                                              running_costs_7_5_tonne_LGV)
    truck_costs['18_tonne_running_costs'] = (truck_costs['18-tonne HGV (24)']*
                                             running_costs_18_tonne_HGV)

    # Adding a column to store the total truck costs for each cluster
    truck_costs['Total_costs_month'] = truck_costs.iloc[:,5:10].sum(axis=1)

    # Removing columns we don't need
    truck_costs = truck_costs[['Pallets', 'Total_costs_month']]
```

```
    # Now working out the total pallets and delivery costs for each cluster

    # Checking the total number of pallets delivered currently and delivery costs spent, per cluster
    pallet_cost_now = (kite_nonDX.groupby('Cluster Label')[['Pallets',
                                                            'DeliveryCost']].sum())

    # Scale the pallet and cost numbers to a business month.
    # Divide the cost numbers by 18 (number of working days in the data)
    # and multiply by 22 (number of working days in a month)
    pallet_cost_now['Pallets'] = pallet_cost_now['Pallets']/18*22
    pallet_cost_now['DeliveryCost'] = pallet_cost_now['DeliveryCost']/18*22


    # Joining the two DataFrames (truck_costs and pallet_cost_now)
    df_cost_savings = pallet_cost_now.join(truck_costs, how = 'inner', rsuffix = '_right')

    # Removing unnecessary columns
    df_cost_savings = df_cost_savings.drop(columns = 'Pallets_right')

    # Calculating the courier cost per pallet
    df_cost_savings['Courier Cost per Pallet (Current)'] = (df_cost_savings['DeliveryCost']/
                                                            df_cost_savings['Pallets'])

    # Concatenating df_cost_savings and Pallets_truck_month_df
    df_cost_savings = pd.concat([df_cost_savings, Pallets_truck_month_df], axis=1)

    # Renaming columns
    df_cost_savings.rename(columns={'DeliveryCost': 'Courier Cost (Current)',
                    'Total_costs_month': 'in-house trucks cost',
                    'Pallets_Month':'Pallets delivered by in-house trucks'}, inplace=True)

    # Calculating the remaining pallets left to deliver by third-party couriers
    df_cost_savings['Remaining Pallets'] = (df_cost_savings['Pallets']-
                                            df_cost_savings['Pallets delivered by in-house trucks'])
    # Calculating the cost to deliver these remaining pallets with third-party couriers
    df_cost_savings['Courier Cost Remaining Pallets'] = \
                                        (df_cost_savings['Courier Cost per Pallet (Current)']*
                                        df_cost_savings['Remaining Pallets'])
    # Calculating the total cost of implementing this partial replacement model
    df_cost_savings['Partial Replacement Model Total Cost'] = \
                                        (df_cost_savings['in-house trucks cost']+
                                        df_cost_savings['Courier Cost Remaining Pallets'])

    # Calcuate the % of total pallets delivered using in-house trucks
    df_cost_savings['Pallets delivered by in-house trucks, % total'] = \
                                        (df_cost_savings['Pallets delivered by in-house trucks']/
                                        df_cost_savings['Pallets']*100)

    # Calculate cost saving £, note this is for the calendar month
    df_cost_savings['cost saving £'] = (df_cost_savings['Courier Cost (Current)'] -
                                        df_cost_savings['Partial Replacement Model Total Cost'])
    df_cost_savings['cost saving %'] = (df_cost_savings['cost saving £']/
                                        df_cost_savings['Courier Cost (Current)'])*100

    return df_cost_savings.round()
```

Source: Analytical Avengers.

Using outputs from the truck allocation function above, this enhanced function produced a cost-saving model based on filling allocated trucks for minimum daily pallets (Model 1).

Figure 22: Model 1: Cost savings for filled minimum daily pallets.

Cluster Label	Pallets	Courier Cost (Current)	in-house trucks cost	Courier Cost per Pallet (Current)	Pallets delivered by in-house trucks	Remaining Pallets	Courier Cost Remaining Pallets	Partial Replacement Model Total Cost	Pallets delivered by in-house trucks, % total	cost saving £	cost saving %
Worcester / Gloucester	199	9,456	4,450	47	194	5	258	4,708	97	4,748	50
Birmingham	879	37,955	5,400	43	522	357	15,415	20,815	59	17,140	45
Nottingham / Leicester	642	30,324	5,400	47	511	132	6,226	11,626	79	18,697	62
Coventry / Oxford	472	19,892	4,450	42	296	176	7,417	11,867	63	8,025	40
Milton Keynes	371	15,854	4,450	43	290	81	3,465	7,915	78	7,939	50

Source: Analytical Avengers.

As stated previously, we then expanded our analysis to investigate the potential cost savings for filling allocated trucks for median daily pallets (Model 2).

Figure 23: Model 2: Cost savings for filled median daily pallets.

Cluster Label	Pallets	Courier Cost (Current)	in-house trucks cost	Courier Cost per Pallet (Current)	Pallets delivered by in-house trucks	Remaining Pallets	Courier Cost Remaining Pallets	Partial Replacement Model Total Cost	Pallets delivered by in-house trucks, % total	cost saving £	cost saving %
Worcester / Gloucester	199	9,456	4,450	47	194	5	258	4,708	97	4,748	50
Birmingham	879	37,955	9,850	43	745	134	5,801	15,651	85	22,304	59
Nottingham / Leicester	642	30,324	8,900	47	565	77	3,651	12,551	88	17,773	59
Coventry / Oxford	472	19,892	5,400	42	430	43	1,795	7,195	91	12,697	64
Milton Keynes	371	15,854	5,400	43	366	6	239	5,639	98	10,215	64

Source: Analytical Avengers.

3.4. Sentiment Analysis

NLP sentiment analysis was conducted to gain deeper insights into delivery issues. We scraped the last 18 months of customer reviews from Trustpilot.

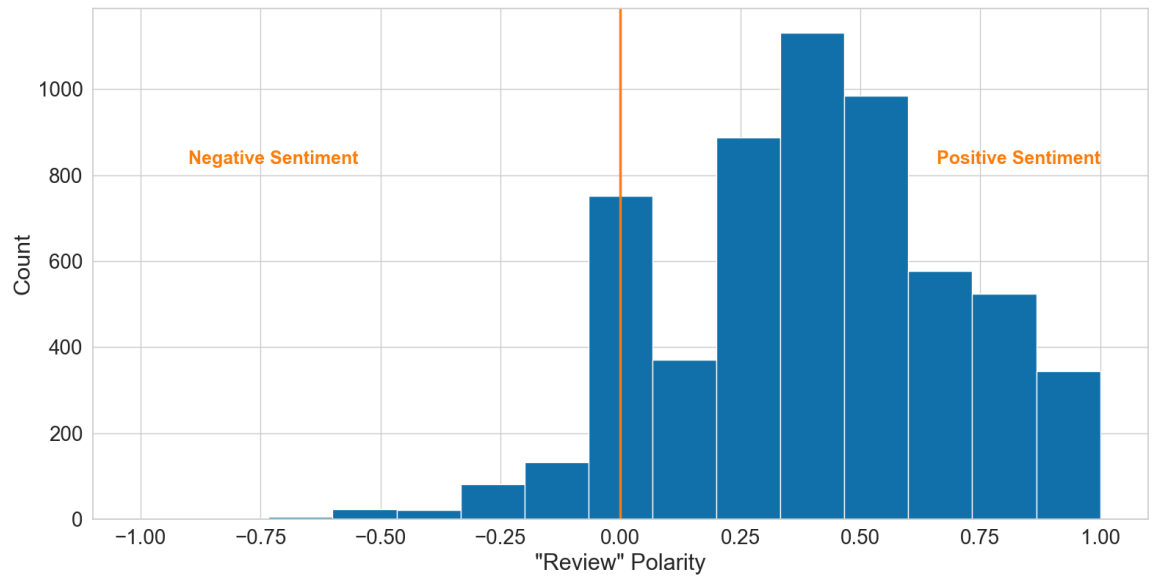
Figure 24: Code for scraping Trustpilot data.

```
from time import sleep
import requests

import pandas as pd
from bs4 import BeautifulSoup

def soup2list(src, list_, attr=None):
    if attr:
        for val in src:
            list_.append(val[attr])
    else:
        for val in src:
            list_.append(val.get_text())

users = []
userReviewNum = []
ratings = []
locations = []
dates = []
reviews = []

from_page = 1
to_page = 300
company = 'kitepackaging'

for i in range(from_page, to_page+1):

   result = requests.get(fr"https://www.trustpilot.com/review/www.kitepackaging.co.uk?page={i}")
   soup = BeautifulSoup(result.content)

   # Trust Pilot was setup in a way that's not friendly to scraping, so this hacky method will do.
   soup2list(soup.find_all('span',
                           {'class','typography_heading-xxs__QKBS8 typography_appearance-default__AAY17'}), users)
   soup2list(soup.find_all('div',
                           {'class','typography_body-m__xgxZ_ typography_appearance-subtle__8_H2l styles_detailsIcon__Fo_ua'}),
             locations)
   soup2list(soup.find_all('span',
                           {'class','typography_body-m__xgxZ_ typography_appearance-subtle__8_H2l'}),
             userReviewNum)
   soup2list(soup.find_all('div',
                           {'class','styles_reviewHeader__iU9Px'}),
             dates)
   soup2list(soup.find_all('div',
                           {'class','styles_reviewHeader__iU9Px'}),
             ratings, attr='data-service-review-rating')
   soup2list(soup.find_all('div',
                           {'class','styles_reviewContent__0Q2Tg'}),
             reviews)

   # To avoid throttling
   sleep(1)

# Create a DataFrame from the extracted data
review_data = pd.DataFrame({
    'User': users, 'Review Number': userReviewNum,
    'Rating': ratings, 'Location': locations,
    'Date': dates, 'Review': reviews})
```

Source: Analytical Avengers.

After data pre-processing[7], frequency distributions of relevant words were explored and visualised with word clouds. In addition, we employed pre-existing polarity and subjectivity scores[8] to assess overall sentiment and identify positive and negative trends.

Figure 25: Polarity scores for Trustpilot reviews.



Source: Trustpilot, Analytical Avengers.

Filtering for one-star reviews and searching for keywords revealed specific pain points consistently contributing to negative feedback.

Figure 26: Code for filtering reviews for keywords and phrases.

```
import re

# Define the updated list of keywords and phrases.
keywords = ['delivery', 'customer service', 'packaging', 'damage', 'late']

# Function to check if a review contains any of the keywords or phrases.
def contains_keywords(review):
    for keyword in keywords:
        if re.search(r'\b{}\b'.format(re.escape(keyword)), review, re.IGNORECASE):
            return True
    return False

# Filter the "poor" reviews based on the keywords or phrases.
filtered_reviews_poorest = poorest_reviews[poorest_reviews['Review'].apply(contains_keywords)]

# Calculate the percentage of 1 and 2 star reviews that contain each keyword.
keyword_percentages = {}
for keyword in keywords:
    total_reviews = len(poorest_reviews)
    keyword_reviews = len(filtered_reviews_poorest[filtered_reviews_poorest['Review'].\
                                    str.contains(keyword, case=False)])
    percentage = (keyword_reviews / total_reviews) * 100
    keyword_percentages[keyword] = percentage

# Display the percentage for each keyword.
for keyword, percentage in keyword_percentages.items():
    print("Percentage of 1 and 2 star reviews containing '{}': {:.2f}%".\
          format(keyword, percentage))
```

Source: Analytical Avengers.

[7] To prepare the text data for NLP (natural language processing) analysis, we employed pre-processing techniques such as converting text to lowercase, removing punctuation, tokenizing, and eliminating stopwords.

[8] Polarity scores indicate the degree of positive or negative sentiment expressed in a text, with values ranging from -1 (strongly negative) to +1 (strongly positive).

Figure 27: Word Cloud of one-star reviews on Trustpilot.

Source: Trustpilot, Analytical Avengers.

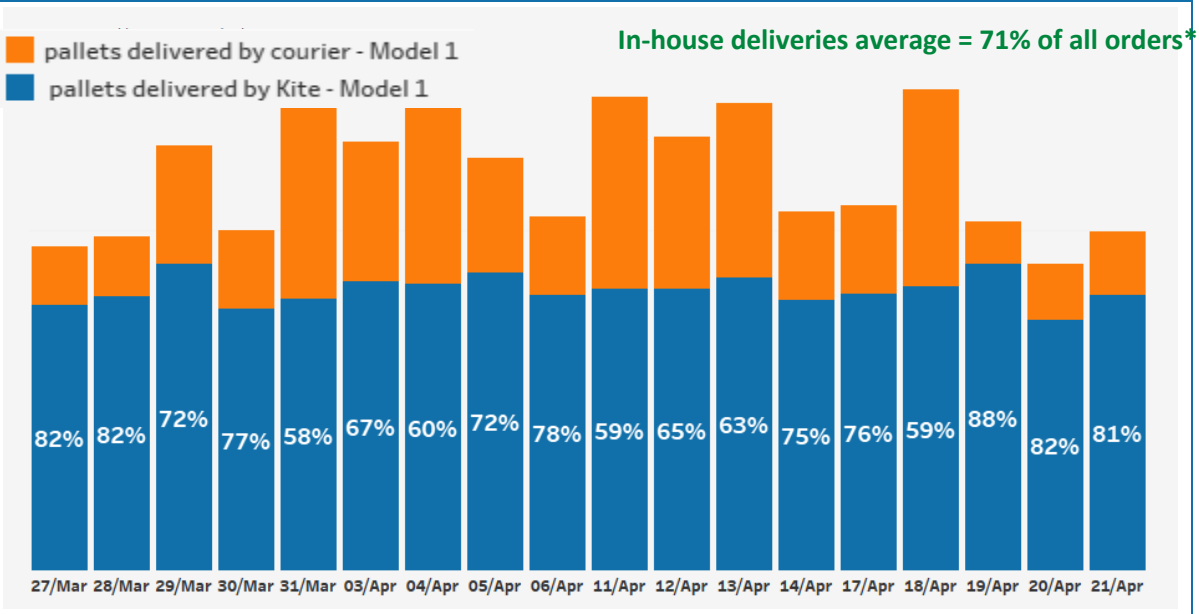
NLP techniques allowed us to gain valuable insights, helping us to appreciate the non-financial benefits of bringing delivery services in-house, such as brand reputation, increased control over delivery processes, and enhanced customer satisfaction.

4. Patterns, Trends, and Insights

4.1. Model 1: Minimum Daily Pallets

Truck allocation was based on minimum daily orders for a month, filling trucks where possible. Five trucks were allocated across the clusters, with a monthly cost of £55,000 and a daily total truck capacity of 90 pallets, taking 71% of orders in-house.

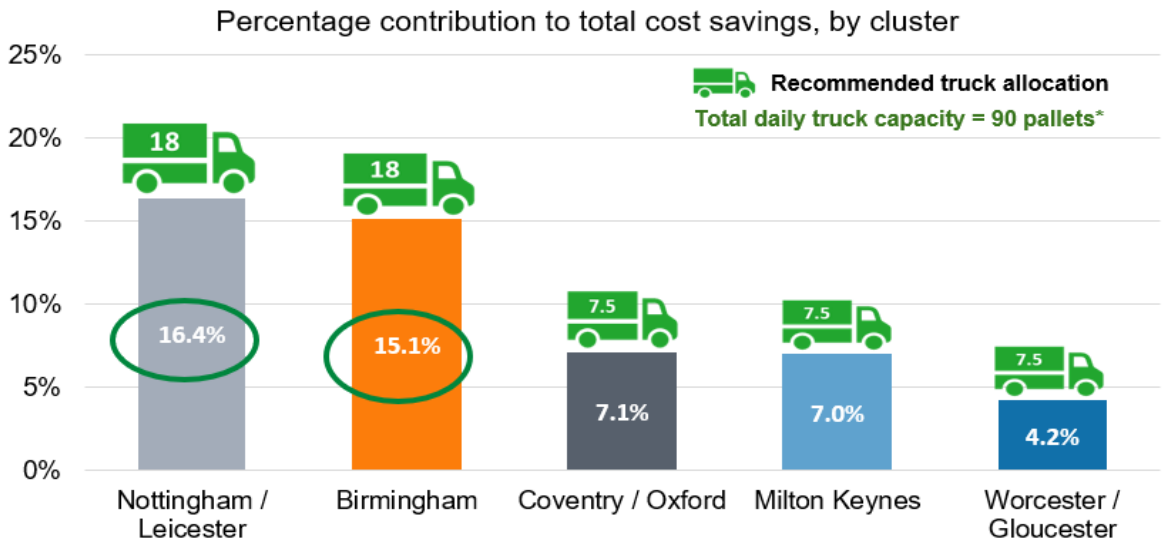
Figure 28: Model 1 results in about 70% of pallets being delivered in-house.



Source: Kite Packaging, Analytical Avengers, * total excludes DX Express.

Looking at cost savings, Nottingham/Leicester and Birmingham clusters could each achieve 15-16% savings, consistent with their high proportion of orders/delivery costs.

Figure 29: Model 1 allocates five trucks and achieves savings across the clusters.

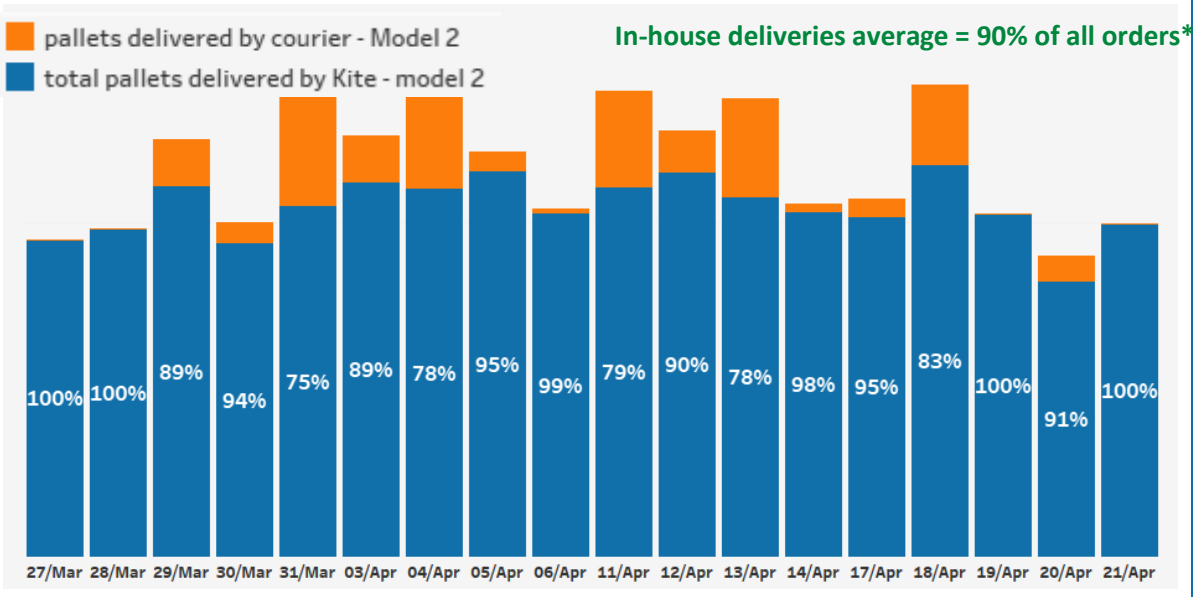


Source: Kite Packaging, Analytical Avengers; * Truck capacity data supplied by Kite.
Note: Based on total monthly delivery costs of £113,481 (raw data scaled to a full business month).

4.2. Model 2: Median Daily Pallets

Truck allocation is based on median daily pallets to determine whether the increased cost of a bigger fleet is offset by cost-savings of bringing a greater proportion of deliveries in-house. The model shows four out of five clusters requiring extra capacity, with 90% of orders delivered in-house.

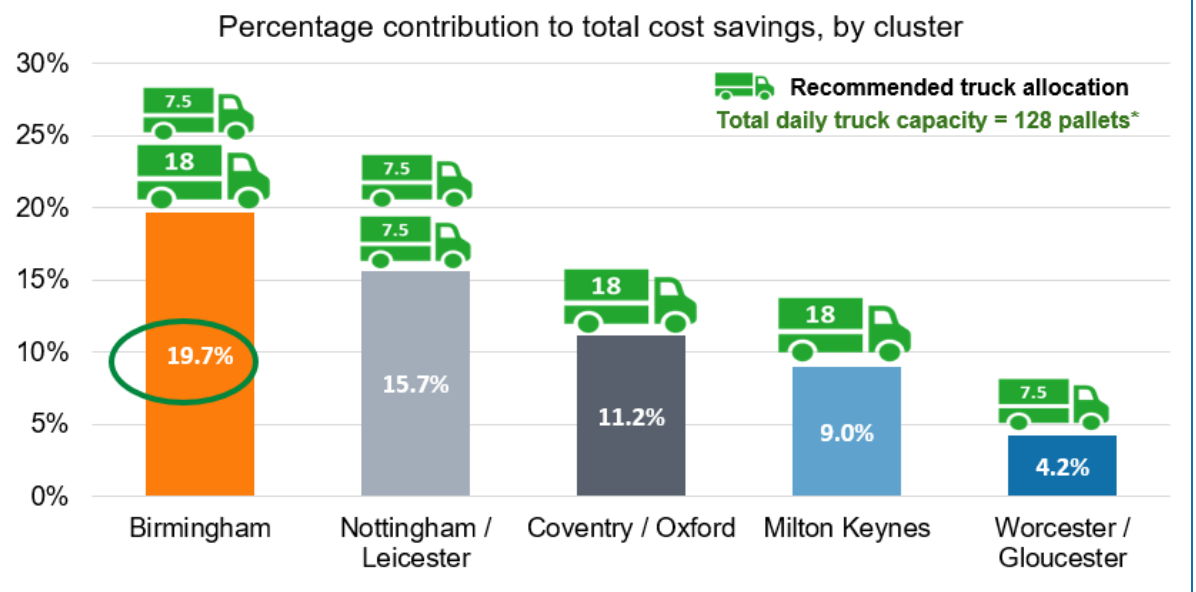
Figure 30: Model 2 results in 90% of pallets being delivered in-house.



Source: Kite Packaging, Analytical Avengers; * total excludes DX Express.

This represents an additional monthly fleet investment but increases cost savings in three of the five clusters. In Birmingham alone cost saving jumps to 20%, fulfilling Kite's need to save 20% on delivery costs around the Coventry distribution centre.

Figure 31: Model 2 shows that increasing the fleet size leads to higher cost savings.

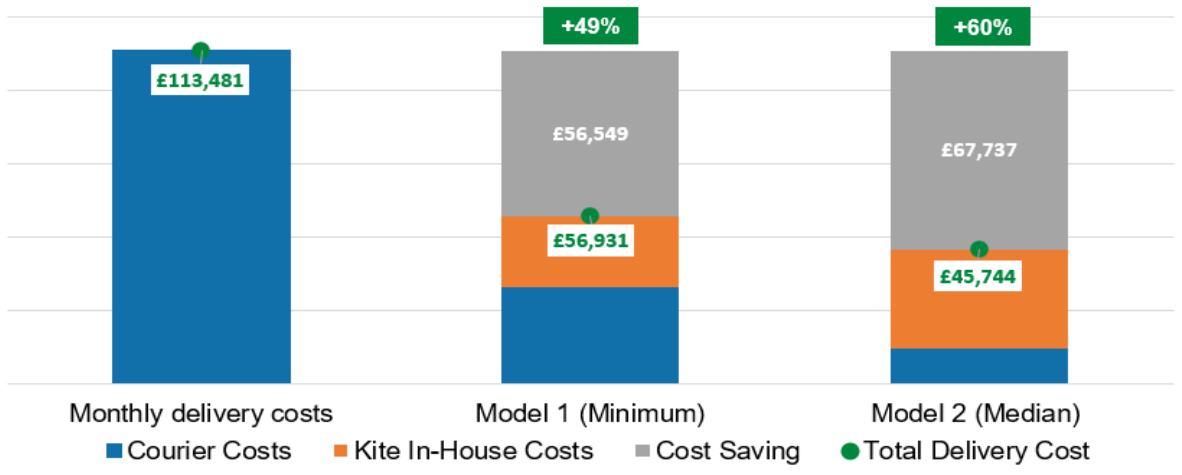


Source: Kite Packaging, Analytical Avengers; * truck capacity data supplied by Kite.
Note: Based on total monthly delivery costs of £113,481 (raw data scaled to a full business month).

4.3. Total Potential Cost Savings

Kite currently spends over £110,000 a month on delivery costs. Model 1, the least financial risk/commitment, saves around £55,000, or almost 50%, within 50 miles of Coventry. By committing to the increased fleet (Model 2), potential savings are even greater, saving nearly £70,000, or 60%.

Figure 32: Total potential savings of 60% on delivery costs using median pallets

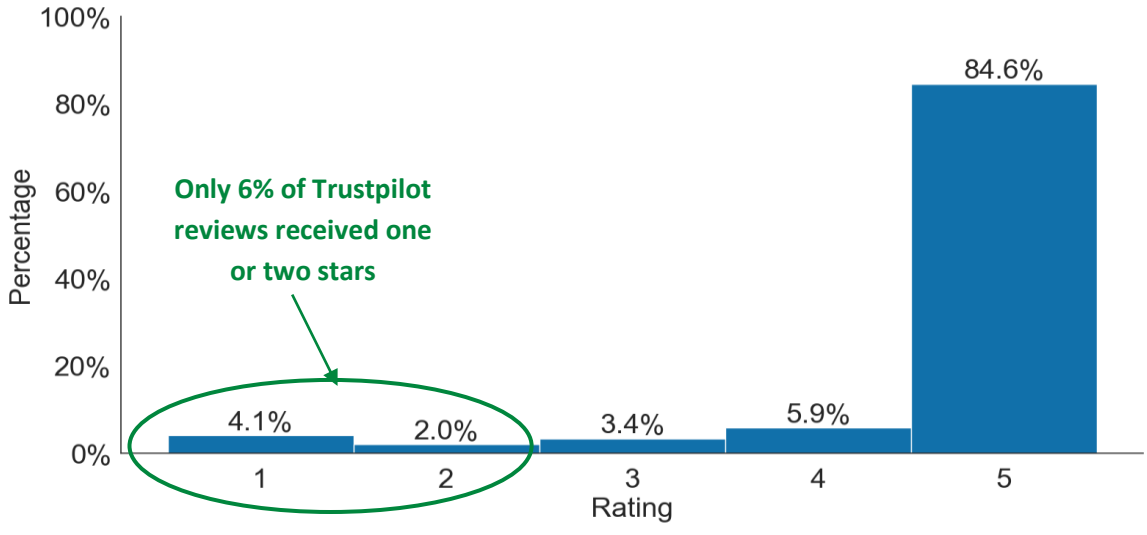


Source: Kite Packaging, Analytical Avengers.
Note: Based on total monthly delivery costs of £113,481 (raw data scaled to a full business month).

4.4. Non-Financial Benefits

Besides increased profitability, there are many other benefits of bringing deliveries in-house, including improved brand and reputation. Managing deliveries ensures product handling and transportation adhere to specific in-house standards, leading to improved customer satisfaction, which drives customer loyalty, repeat business and word of mouth referrals.

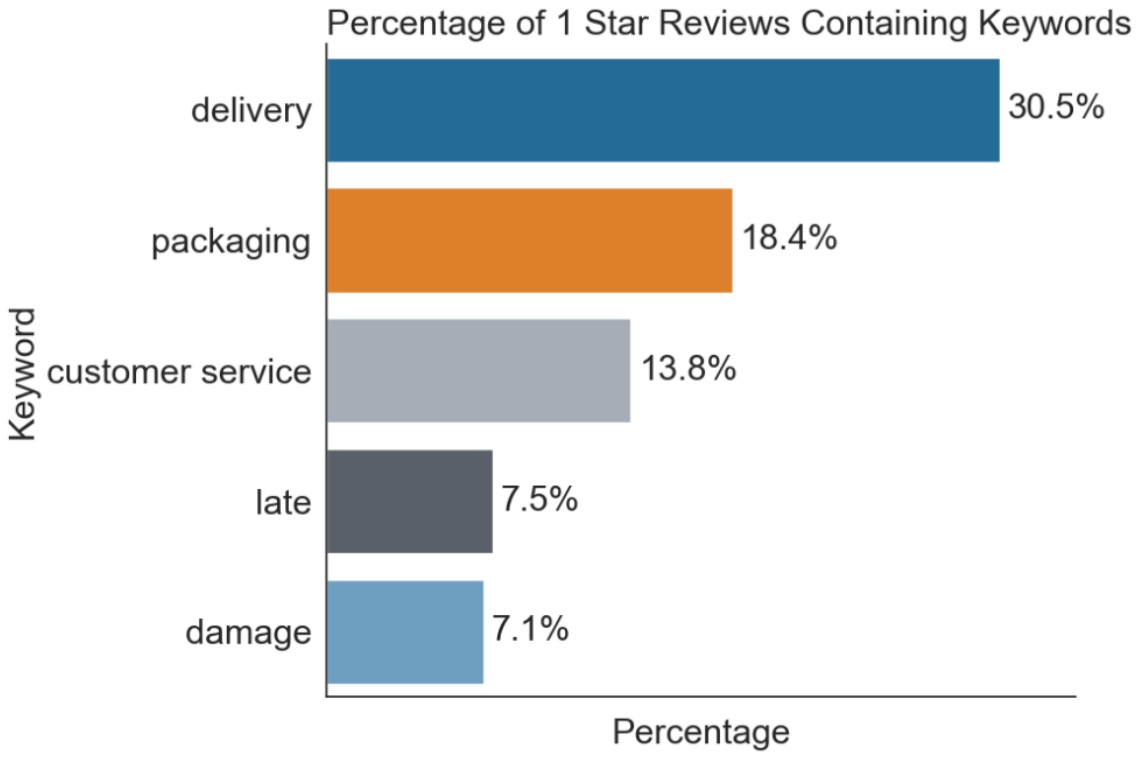
Figure 33: Analysis of Kite's Trustpilot reviews reveals its overall ratings are excellent.



Source: Trustpilot, Analytical Avengers.

While Kite's Trustpilot reviews reveal excellent overall ratings, 'delivery' was one of the most common words when analysing one-star reviews. Other related issues, such as 'late' and 'damage' also appeared in negative reviews. Integrating in-house deliveries will not only save costs but also improve customer experiences.

Figure 34: Delivery issues are a leading cause of low customer ratings.



Source: Trustpilot, Analytical Avengers.

4.5. Recommendations

- Run a pilot study to minimise risks and ensure smooth implementation. Birmingham would be suitable, owing to its high potential for cost savings.
- Replace deliveries gradually across clusters to minimise risk and disruption.
- Determine truck allocation using median pallets, a larger financial commitment but a greater cost saving and also control over a higher proportion of deliveries.
- Investigate the feasibility of a hub and spoke model.
- Consider the impact of route optimisation software to make further cost savings.
- Investigate the potential for investment in electric vehicles.

5. Appendix

Model 1: Total Cost Savings

For all clusters, using filled trucks for **minimum daily pallets** could save Kite **49% in delivery costs per month** around its Coventry distribution centre.

Cluster	Total delivery cost per month	Minimum daily pallets	Monthly minimum pallets for Kite	Minimum trucks required	Trucks cost	Monthly pallets for couriers	Courier cost	Cost savings	% cost saving by cluster	% total cost savings
Worcester / Gloucester	**£9,456**	4.1	194	1 x 7.5 tonne LGV (14 pallets)	£4,450	5	£258	£4,748	**+50%**	**4%**
Birmingham	**£37,955**	19.2	522	1 x 18-tonne HGV (24 pallets)	£5,400	357	£15,415	£17,140	**+45%**	**15%**
Nottingham / Leicester	**£30,324**	18.1	511	1 x 18-tonne HGV	£5,400	132	£6,226	£18,697	**+62%**	**16%**
Coventry / Oxford	**£19,892**	9.4	296	1 x 7.5 tonne LGV	£4,450	176	£7,417	£8,025	**+40%**	**7%**
Milton Keynes	**£15,854**	9.0	290	1 x 7.5 tonne LGV	£4,450	81	£3,465	£7,939	**+50%**	**7%**
Total	£113,481	-	-	-	£24,150	-	£32,781	£56,549	-	49%

Note: Figures taken from model results which used rounded up values.

Model 2: Total Cost Savings

For all clusters, using filled trucks for **median daily pallets** could save Kite **60% in delivery costs per month** around its Coventry distribution centre.

Cluster	Total delivery cost per month	Median daily pallets	Monthly median pallets for Kite	Minimum trucks required	Trucks cost	Monthly pallets for couriers	Courier cost	Cost savings	% cost saving by cluster	% total cost savings
Worcester / Gloucester	**£9,456**	8.5	194	1 x 7.5 tonne LGV	£4,450	5	£258	£4,748	**+50%**	**4%**
Birmingham	**£37,955**	36.9	745	1 x 18-tonne HGV; 1 x 7.5 tonne LGV	£9,850	134	£5,801	£22,304	**+59%**	**20%**
Nottingham / Leicester	**£30,324**	27.4	565	2 x 7.5 tonne LGV	£8,900	77	£3,651	£17,773	**+59%**	**16%**
Coventry / Oxford	**£19,892**	20.9	430	1 x 18-tonne HGV	£5,400	43	£1,795	£12,697	**+64%**	**11%**
Milton Keynes	**£15,854**	16.2	366	1 x 18-tonne HGV	£5,400	6	£239	£10,215	**+64%**	**9%**
Total	**£113,481**	-	-	-	**£34,000**	-	**£11,744**	**£67,737**	-	**60%**

Note: Figures taken from model results which used rounded up values.