Supply Chain Dilemma

Gala Groceries is a technology-led grocery store chain based in the USA. They rely heavily on new technologies, such as IoT to give them a competitive edge over other grocery stores. Groceries are highly perishable items. If you overstock, you are wasting money on excessive storage and waste, but if you understock, then you risk losing customers.

This is logistic regression model built for Gala Groceries , a technology-led grocery store chain based in the USA to help them know and predict how to better stock grocery items that they sell

---

Task 1 - Exploratory Data Analysis

Section 1 - Importing Modules

In [2]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

Section 2 - Data loading using Pandas

Loading `sample_sales_data.csv` dataset so that we can work with them in Python. For this notebook and all further notebooks, it will be assumed that the CSV files will the placed in the same file location as the notebook. If they are not, please adjust the directory within the read_csv method accordingly.

In [3]:
```
sales_data = pd.read_csv(r"C:\Users\Mr.Hassan\DataspellProjects\Gala Foods\sample_sales_data.csv")
```

Section 3 - Descriptive statistics

In this section, we try to gain a description of the data, that is: what columns are present, how many null values exist and what data types exists within each column.

To get started,this is an explanation of what the column names mean

- transaction_id = this is a unique ID that is assigned to each transaction
- timestamp = this is the datetime at which the transaction was made
- product_id = this is an ID that is assigned to the product that was sold. Each product has a unique ID
- category = this is the category that the product is contained within
- customer_type = this is the type of customer that made the transaction
- unit_price = the price that 1 unit of this item sells for
- quantity = the number of units sold for this product within this transaction
- total = the total amount payable by the customer
- payment_type = the payment method used by the customer

Data Types

In [4]:
```
sales_data.head()
```

Out[4]:

	Unnamed: 0	transaction_id	timestamp	product_id	category	customer_type	unit_price	quantity	total	payment_type
0	0	a1c82654-c52c-45b3-8ce8-4c2a1efe63ed	2022-03-02 09:51:38	3bc6c1ea-0198-46de-9ffd-514ae3338713	fruit	gold	3.99	2	7.98	e-wallet
1	1	931ad550-09e8-4da6-beaa-8c9d17be9c60	2022-03-06 10:33:59	ad81b46c-bf38-41cf-9b54-5fe7f5eba93e	fruit	standard	3.99	1	3.99	e-wallet
2	2	ae133534-6f61-4cd6-b6b8-d1c1d8d90aea	2022-03-04 17:20:21	7c55cbd4-f306-4c04-a030-628cbe7867c1	fruit	premium	0.19	2	0.38	e-wallet
3	3	157cebd9-aaf0-475d-8a11-7c8e0f5b76e4	2022-03-02 17:23:58	80da8348-1707-403f-8be7-9e6deeccc883	fruit	gold	0.19	4	0.76	e-wallet
4	4	a81a6cd3-5e0c-44a2-826c-aea43e46c514	2022-03-05 14:32:43	7f5e86e6-f06f-45f6-bf44-27b095c9ad1d	fruit	basic	4.49	2	8.98	debit card

In [5]:

```
sales_data.info()
#There are no null values in this data set.
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7829 entries, 0 to 7828
Data columns (total 10 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Unnamed: 0      7829 non-null   int64  
 1   transaction_id  7829 non-null   object 
 2   timestamp       7829 non-null   object 
 3   product_id      7829 non-null   object 
 4   category        7829 non-null   object 
 5   customer_type   7829 non-null   object 
 6   unit_price      7829 non-null   float64
 7   quantity        7829 non-null   int64  
 8   total           7829 non-null   float64
 9   payment_type    7829 non-null   object 
dtypes: float64(2), int64(2), object(6)
memory usage: 611.8+ KB
```

Statistics

In [6]:

```
sales_data.describe()
#We might have skewed data columns here
```

Out[6]:

	Unnamed: 0	unit_price	quantity	total
count	7829.000000	7829.000000	7829.000000	7829.000000
mean	3914.000000	7.819480	2.501597	19.709905
std	2260.181962	5.388088	1.122722	17.446680
min	0.000000	0.190000	1.000000	0.190000
25%	1957.000000	3.990000	1.000000	6.570000
50%	3914.000000	7.190000	3.000000	14.970000
75%	5871.000000	11.190000	4.000000	28.470000
max	7828.000000	23.990000	4.000000	95.960000

Section 4 - Visualisation

Now that we have computed some descriptive statistics of the dataset, let's create some visualisations.

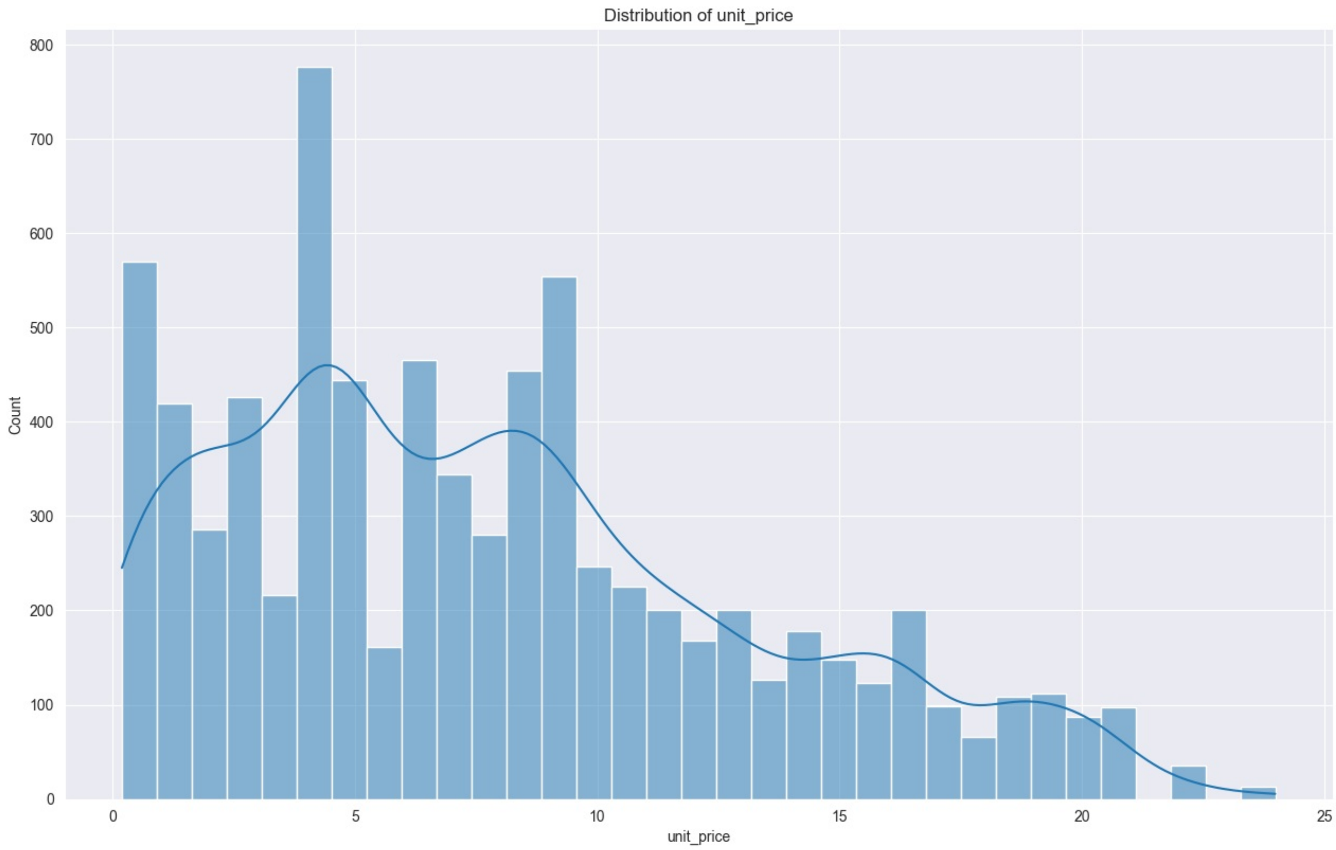
In [7]:

```
def plot_continuous_distribution(data: pd.DataFrame = None, column: str = None, height: int = 8):
    sns.displot(data, x=column, kde=True, height=height, aspect=height/5).set(title=f'Distribution of {column}'

def plot_categorical_distribution(data: pd.DataFrame = None, column: str = None, height: int = 8, aspect: int =
    sns.catplot(data=data, x=column, kind='count', height=height, aspect=aspect,order=data[column].value_counts

def correlation_plot(data: pd.DataFrame = None):
    corr = data.corr()
    corr.style.background_gradient(cmap='coolwarm')
    sns.heatmap(corr, xticklabels=corr.columns.values, yticklabels=corr.columns.values, annot = True, annot_kws
    # Axis ticks size
    plt.xticks(fontsize=10)
    plt.yticks(fontsize=10)
    plt.show()
```

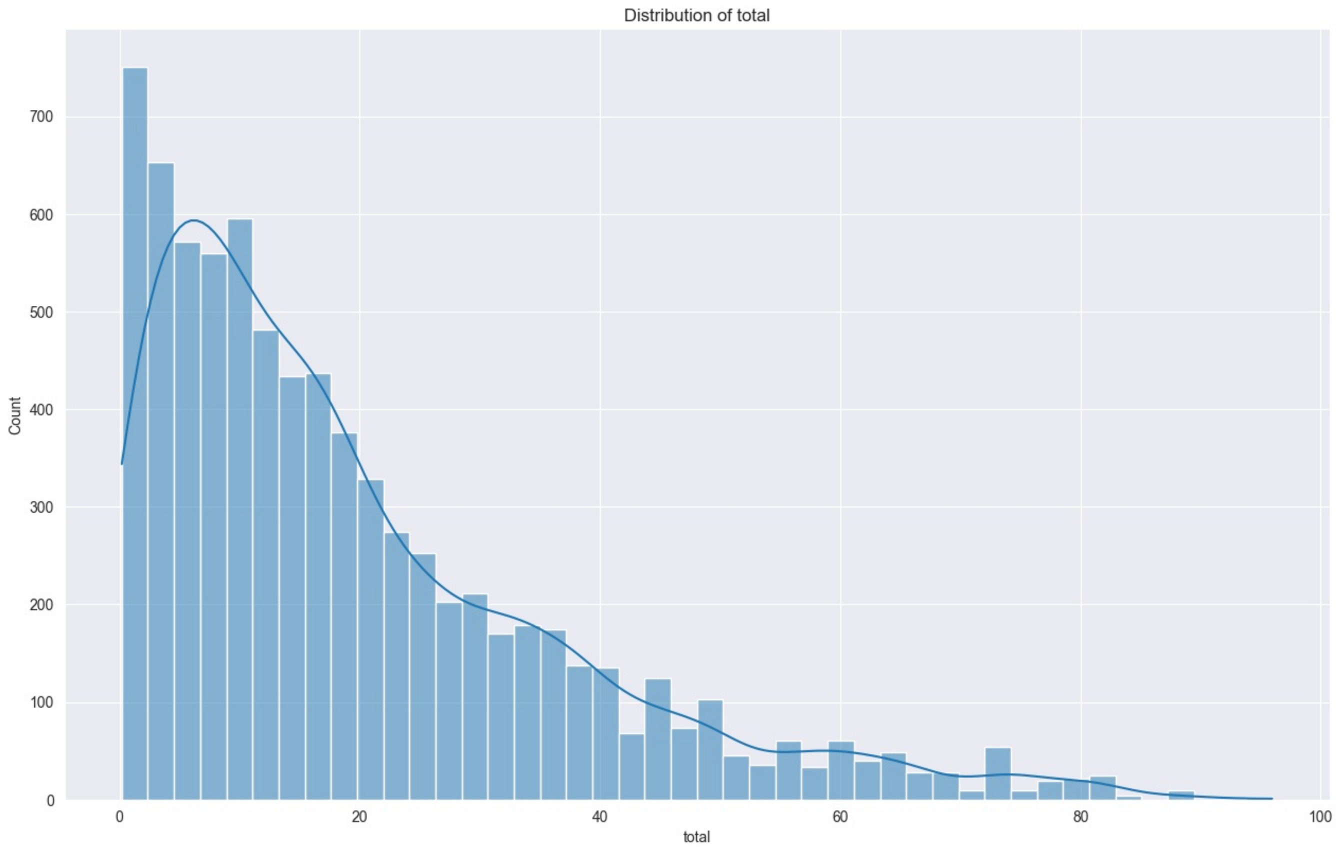
To analyse the dataset, below are snippets of helper functions to visualise different columns within the dataset.

- plot_continuous_distribution = this is to visualise the distribution of numeric columns

In [8]:
```
plot_continuous_distribution(sales_data,"unit_price")
```

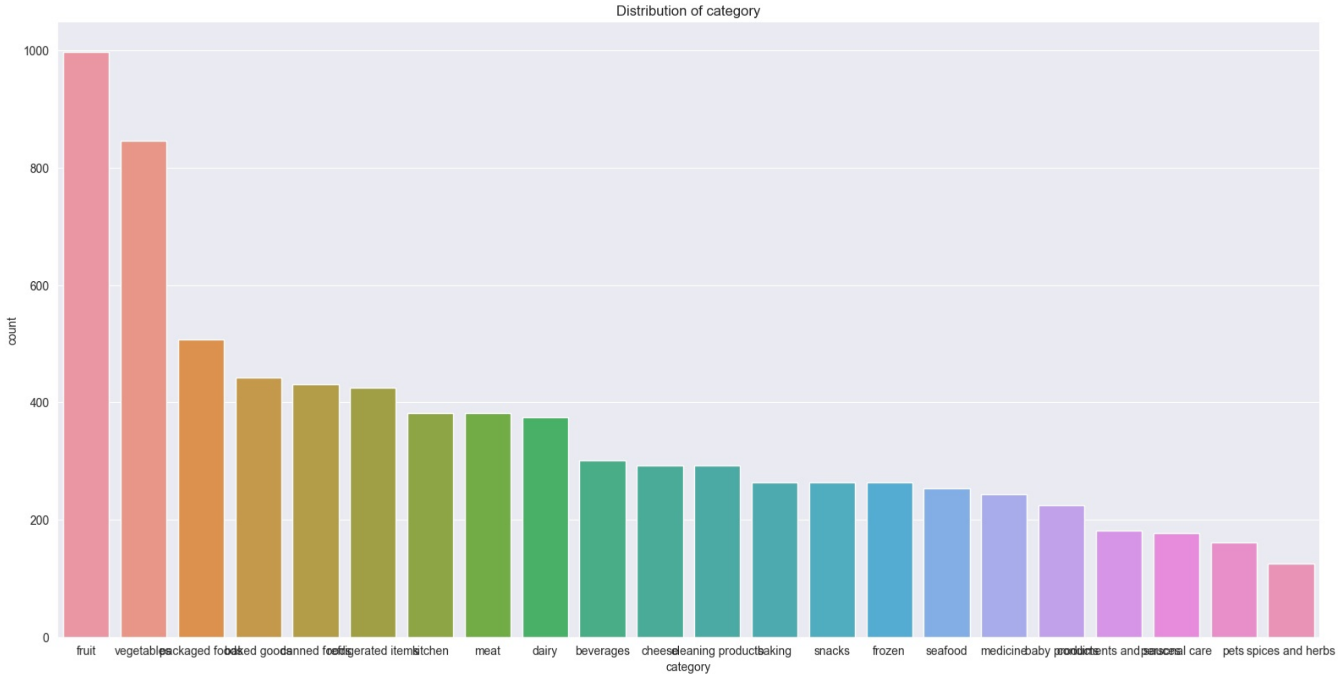


In [9]:
```
plot_continuous_distribution(sales_data,"total")
```

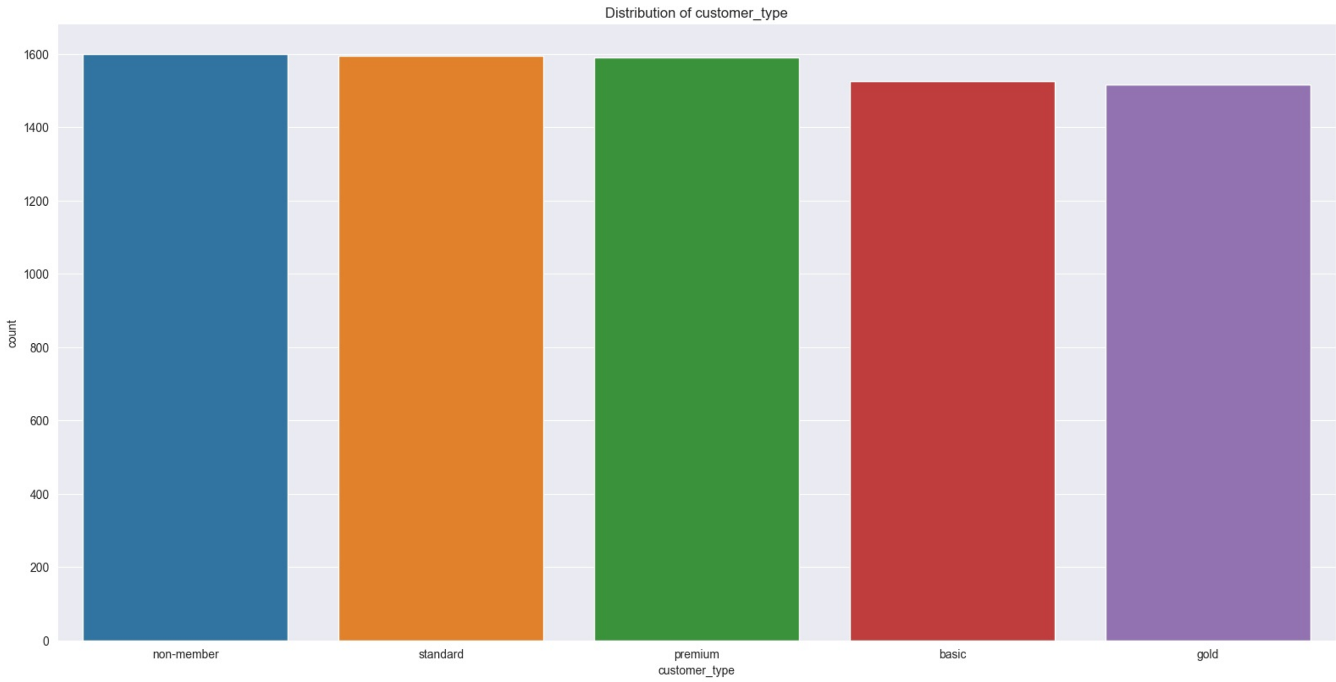


- plot_categorical_distribution = this is to visualise the distribution of categorical columns

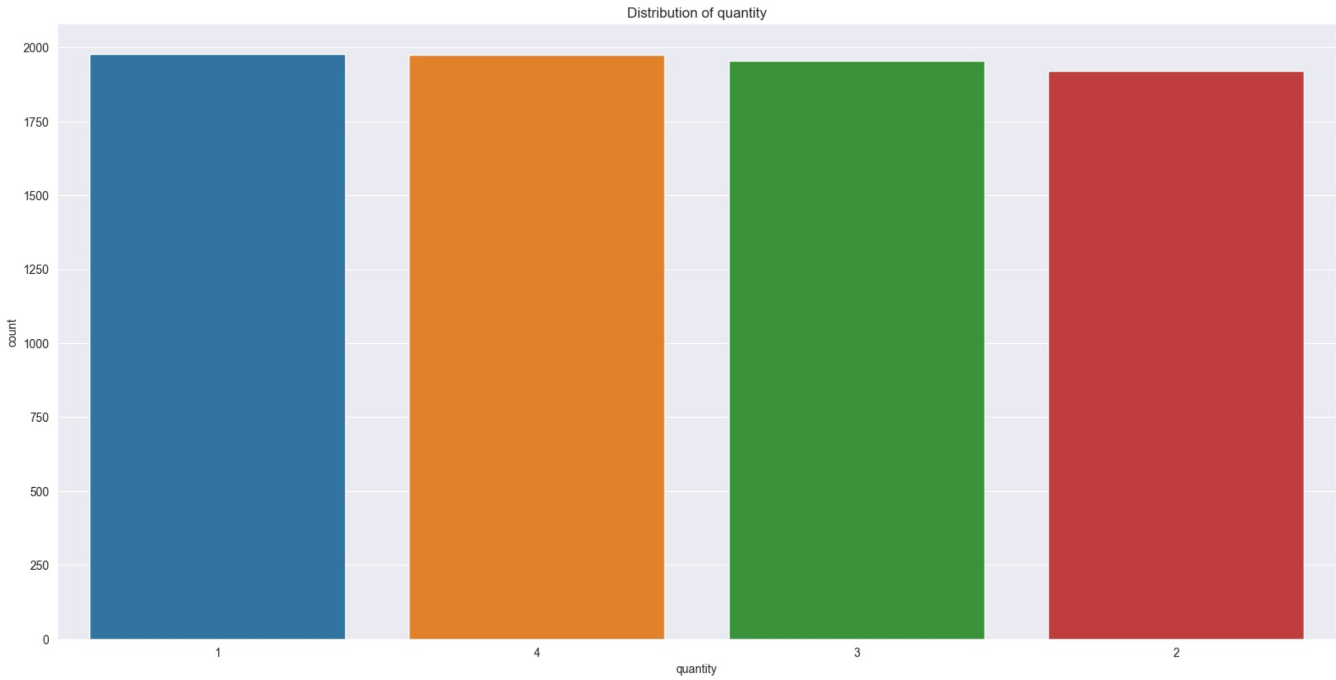
In [10]:
```
plot_categorical_distribution(sales_data,"category")
```



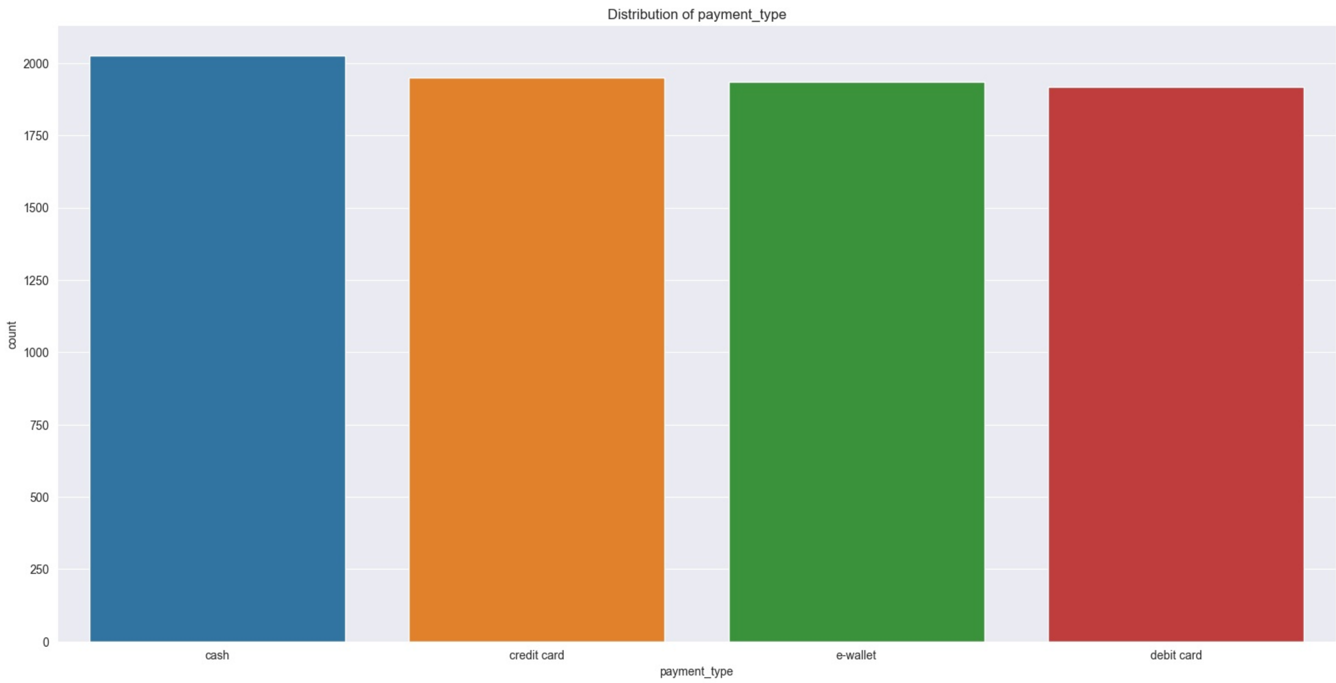

```
In [11]: plot_categorical_distribution(sales_data,"customer_type")
```




```
In [12]: plot_categorical_distribution(sales_data,"quantity")
```

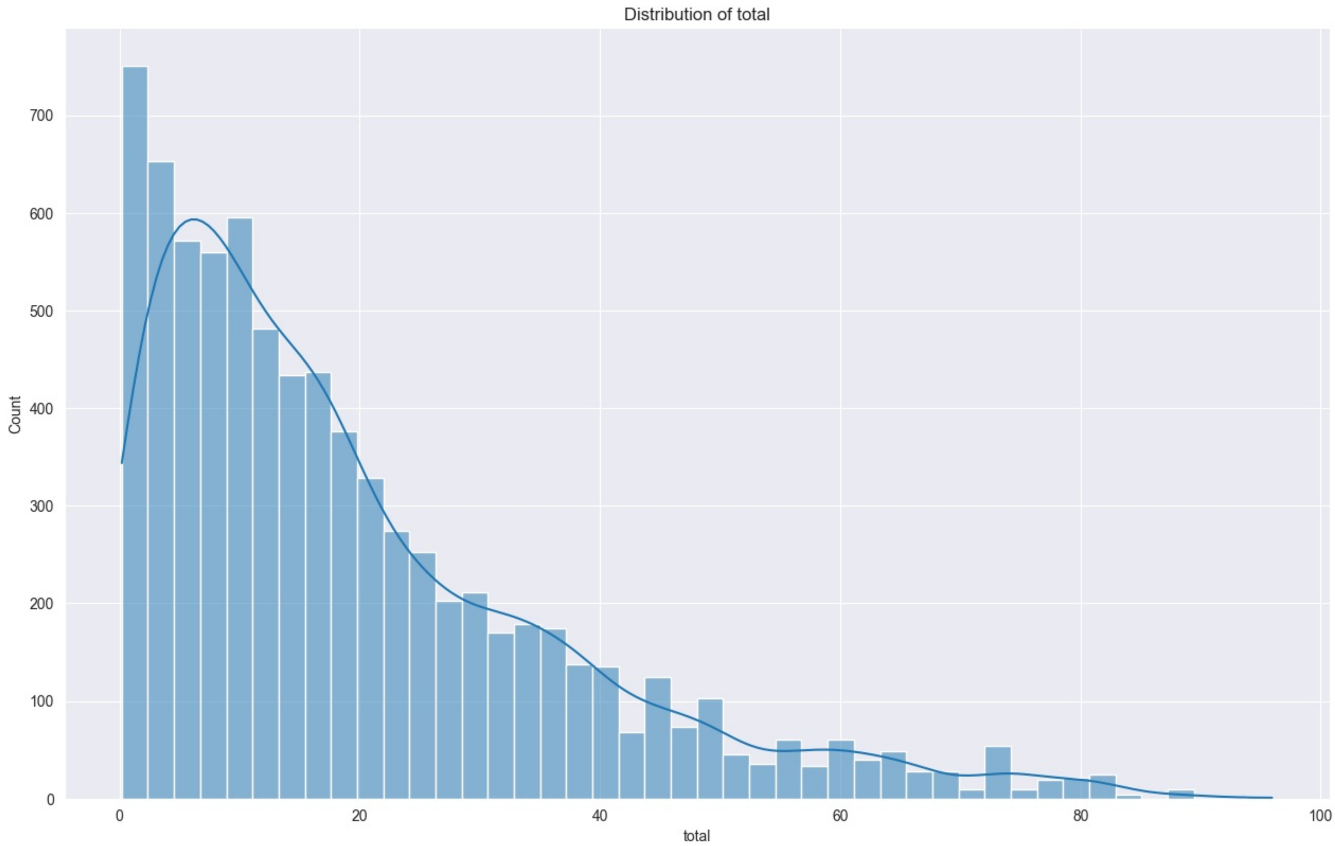


In [13]:
```
plot_categorical_distribution(sales_data,"payment_type")
```

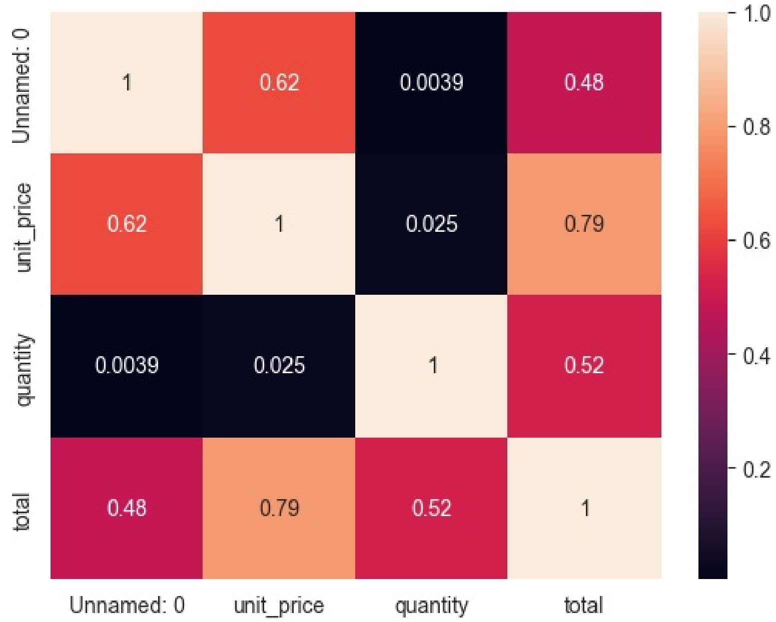


- correlation_plot = this is to plot the correlations between the numeric columns within the data

In [14]:
```
plot_continuous_distribution(sales_data,"total")
```



In [15]:
```
correlation_plot(sales_data)
```



Section 5 - Summary

We have completed an initial exploratory data analysis on the sample of data provided. We should now have a solid understanding of the data. I found the following insights as part of the analysis:

- Fruit & vegetables are the 2 most frequently bought product categories
- Non-members are the most frequent buyers within the store
- Cash is the most frequently used payment method

`DISCREPANCY`

The client wants to know `"How to better stock the items that they sell"` From this dataset, it is impossible to answer that question. In order to make the next step on this project with the client, it is clear that:

- We need more rows of data. The current sample is only from 1 store and 1 week worth of data
- We need to frame the specific problem statement that we want to solve. The current business problem is too broad, we should

narrow down the focus in order to deliver a valuable end product
- We need more features. Based on the problem statement that we move forward with, we need more columns (features) that may help us to understand the outcome that we're solving for

Cont. from Task 1(Exploratory Data Analysis)

Task 2 - Modeling

Section 6 - Importing Modules to be used

In [16]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

Section 7 - Data loading

Similar to before, let's load our data from the 3 datasets provided Loading `sample_sales_data.csv` , `sensor_stock_levels` , `sensor_storage_temperature` dataset so that we can work with them in Python. For this notebook and all further notebooks, it will be assumed that the CSV files will the placed in the same file location as the notebook. If they are not, please adjust the directory within the read_csv method accordingly.

In [17]:
```
sales_data = pd.read_csv(r"C:\Users\Mr.Hassan\DataspellProjects\Gala Foods\sales.csv")
sensor_storage_temperature = pd.read_csv(r"C:\Users\Mr.Hassan\DataspellProjects\Gala Foods\sensor_storage_tempe
sensor_stock_levels = pd.read_csv(r"C:\Users\Mr.Hassan\DataspellProjects\Gala Foods\sensor_stock_levels.csv")
```

In [18]:
```
sales_data.drop(columns=["Unnamed: 0"],inplace=True)
sales_data
```

Out[18]:

	transaction_id	timestamp	product_id	category	customer_type	unit_price	quantity	total	payment_type
0	a1c82654-c52c-45b3-8ce8-4c2a1efe63ed	2022-03-02 09:51:38	3bc6c1ea-0198-46de-9ffd-514ae3338713	fruit	gold	3.99	2	7.98	e-wallet
1	931ad550-09e8-4da6-beaa-8c9d17be9c60	2022-03-06 10:33:59	ad81b46c-bf38-41cf-9b54-5fe7f5eba93e	fruit	standard	3.99	1	3.99	e-wallet
2	ae133534-6f61-4cd6-b6b8-d1c1d8d90aea	2022-03-04 17:20:21	7c55cbd4-f306-4c04-a030-628cbe7867c1	fruit	premium	0.19	2	0.38	e-wallet
3	157cebd9-aaf0-475d-8a11-7c8e0f5b76e4	2022-03-02 17:23:58	80da8348-1707-403f-8be7-9e6deeccc883	fruit	gold	0.19	4	0.76	e-wallet
4	a81a6cd3-5e0c-44a2-826c-aea43e46c514	2022-03-05 14:32:43	7f5e86e6-f06f-45f6-bf44-27b095c9ad1d	fruit	basic	4.49	2	8.98	debit card
...	...	...	...	...	...	...	...	...	...
7824	6c19b9fc-f86d-4526-9dfe-d8027a4d13ee	2022-03-03 18:22:09	bc6187a9-d508-482b-9ca6-590d1cc7524f	cleaning products	basic	14.19	2	28.38	e-wallet
7825	1c69824b-e399-4b79-a5e7-04a3a7db0681	2022-03-04 19:14:46	707e4237-191c-4cc9-85af-383a6c1cb2ab	cleaning products	standard	16.99	1	16.99	credit card
7826	79aee7d6-1405-4345-9a15-92541e9e1e74	2022-03-03 14:00:09	a9325c1a-2715-41df-b7f4-3078fa5ecd97	cleaning products	basic	14.19	2	28.38	credit card
7827	e5cc4f88-e5b7-4ad5-bc1b-12a828a14f55	2022-03-04 15:11:38	707e4237-191c-4cc9-85af-383a6c1cb2ab	cleaning products	basic	16.99	4	67.96	cash
7828	afd70b4f-ee21-402d-8d8f-0d9e13c2bea6	2022-03-06 13:50:36	d6ccd088-11be-4c25-aa1f-ea87c01a04db	cleaning products	non-member	14.99	4	59.96	debit card

7829 rows × 9 columns

In [19]:
```
sensor_stock_levels.drop(columns=["Unnamed: 0"],inplace=True)
sensor_stock_levels
```

Out[19]:

	id	timestamp	product_id	estimated_stock_pct
0	4220e505-c247-478d-9831-6b9f87a4488a	2022-03-07 12:13:02	f658605e-75f3-4fed-a655-c0903f344427	0.75
1	f2612b26-fc82-49ea-8940-0751fdd4d9ef	2022-03-07 16:39:46	de06083a-f5c0-451d-b2f4-9ab88b52609d	0.48
2	989a287f-67e6-4478-aa49-c3a35dac0e2e	2022-03-01 18:17:43	ce8f3a04-d1a4-43b1-a7c2-fa1b8e7674c8	0.58
3	af8e5683-d247-46ac-9909-1a77bdebefb2	2022-03-02 14:29:09	c21e3ba9-92a3-4745-92c2-6faef73223f7	0.79
4	08a32247-3f44-4002-85fb-c198434dd4bb	2022-03-02 13:46:18	7f478817-aa5b-44e9-9059-8045228c9eb0	0.22
...	...	...	...	...
14995	b9bf6788-09f3-490b-959b-dc5b55edb4b6	2022-03-04 10:52:50	e37658de-3649-4ddb-9c73-b868dd69d3fe	0.66
14996	9ff1cc01-020f-491a-bafd-13552dccff44	2022-03-02 12:25:48	fbeb39cc-8cd0-4143-bdfb-77658a02dec9	0.99
14997	4d8101de-e8a2-4af9-9764-7a3a22aa7084	2022-03-03 17:36:44	8e21dcec-d775-4969-8334-05a37a5fd189	0.72
14998	5f2a7b1e-b3c4-4395-8425-c960e22f701d	2022-03-02 19:42:47	9708cf5b-aa69-4320-a013-9d234c40e63f	0.95
14999	af6f4493-e49d-4dcb-951d-308e6cce267b	2022-03-06 17:18:27	3bc6c1ea-0198-46de-9ffd-514ae3338713	0.75

15000 rows × 4 columns

In [20]:

```
sensor_storage_temperature.drop(columns=["Unnamed: 0"],inplace=True)
sensor_storage_temperature
```

Out[20]:

	id	timestamp	temperature
0	d1ca1ef8-0eac-42fc-af80-97106efc7b13	2022-03-07 15:55:20	2.96
1	4b8a66c4-0f3a-4f16-826f-8cf9397e9d18	2022-03-01 09:18:22	1.88
2	3d47a0c7-1e72-4512-812f-b6b5d8428cf3	2022-03-04 15:12:26	1.78
3	9500357b-ce15-424a-837a-7677b386f471	2022-03-02 12:30:42	2.18
4	c4b61fec-99c2-4c6d-8e5d-4edd8c9632fa	2022-03-05 09:09:33	1.38
...	...	...	...
23885	17bcff56-9965-4e9f-ad5f-107f0f3be93f	2022-03-01 10:40:43	-1.46
23886	51d4eb44-04bd-4d6a-b777-0653bc173303	2022-03-05 17:07:49	-19.37
23887	bbcacfc4-3b59-47ee-b9e1-7dd3bd588748	2022-03-01 16:15:41	-2.89
23888	5c4d567b-4bcf-4fcd-86b7-e2db5de6e439	2022-03-07 14:44:52	-2.56
23889	589c28e1-f1f3-4efb-af6d-9f194c4d7d5b	2022-03-01 16:33:41	0.13

23890 rows × 3 columns

Section 8 - Descriptive Statistics

In this section, we try to gain a description of the data, that is: what columns are present, how many null values exist and what data types exists within each column.

- SALES DATA

In [21]:

```
sales_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7829 entries, 0 to 7828
Data columns (total 9 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   transaction_id  7829 non-null   object 
 1   timestamp       7829 non-null   object 
 2   product_id      7829 non-null   object 
 3   category        7829 non-null   object 
 4   customer_type   7829 non-null   object 
 5   unit_price      7829 non-null   float64
 6   quantity        7829 non-null   int64  
 7   total           7829 non-null   float64
 8   payment_type    7829 non-null   object 
dtypes: float64(2), int64(1), object(6)
memory usage: 550.6+ KB
```

In [22]:

```
sales_data.describe()
```

Out[22]:

	unit_price	quantity	total
count	7829.000000	7829.000000	7829.000000
mean	7.819480	2.501597	19.709905
std	5.388088	1.122722	17.446680
min	0.190000	1.000000	0.190000
25%	3.990000	1.000000	6.570000
50%	7.190000	3.000000	14.970000
75%	11.190000	4.000000	28.470000
max	23.990000	4.000000	95.960000

- sensor stock data

In [23]:
```
sensor_stock_levels.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15000 entries, 0 to 14999
Data columns (total 4 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   id                   15000 non-null  object 
 1   timestamp            15000 non-null  object 
 2   product_id           15000 non-null  object 
 3   estimated_stock_pct  15000 non-null  float64
dtypes: float64(1), object(3)
memory usage: 468.9+ KB
```

In [24]:
```
sensor_stock_levels.describe()
```

Out[24]:

	estimated_stock_pct
count	15000.000000
mean	0.502735
std	0.286842
min	0.010000
25%	0.260000
50%	0.500000
75%	0.750000
max	1.000000

- sensor storage temperature

In [25]:
```
sensor_storage_temperature.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 23890 entries, 0 to 23889
Data columns (total 3 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   id           23890 non-null  object 
 1   timestamp    23890 non-null  object 
 2   temperature  23890 non-null  float64
dtypes: float64(1), object(2)
memory usage: 560.0+ KB
```

In [26]:
```
sensor_storage_temperature.describe()
```

Out[26]:

	temperature
count	23890.000000
mean	-0.207075
std	11.217649
min	-30.990000
25%	-2.860000
50%	-1.000000
75%	1.840000
max	34.990000

If we revisit the problem statement: `Can we accurately predict the stock levels of products, based on sales` `data and sensor data, on an hourly basis in order to more intelligently procure products from our suppliers.`

The client indicates that they want the model to predict on an hourly basis. Looking at the data model, we can see that only column that we can use to merge the 3 datasets together is timestamp.

1. we must first transform the timestamp format to datetime format,
2. And then convert these timestamp columns in all 3 datasets to be based on the hour of the day.

In [27]:
```
#A function that converts timestamp to datetime
def convert_to_datetime(data:pd.DataFrame=None, column:str=None):
    data[column] = pd.to_datetime(data[column],format="%Y-%m-%d %H:%M:%S")
```

In [28]:
```
convert_to_datetime(sales_data,"timestamp")
convert_to_datetime(sensor_storage_temperature,"timestamp")
convert_to_datetime(sensor_stock_levels,"timestamp")
```

In [29]:
```
#All timestamps have now been converted into datetimes
sensor_stock_levels.info()
sales_data.info()
sensor_storage_temperature.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15000 entries, 0 to 14999
Data columns (total 4 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   id                   15000 non-null  object
 1   timestamp            15000 non-null  datetime64[ns]
 2   product_id           15000 non-null  object
 3   estimated_stock_pct  15000 non-null  float64
dtypes: datetime64[ns](1), float64(1), object(2)
memory usage: 468.9+ KB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7829 entries, 0 to 7828
Data columns (total 9 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   transaction_id  7829 non-null   object
 1   timestamp       7829 non-null   datetime64[ns]
 2   product_id      7829 non-null   object
 3   category        7829 non-null   object
 4   customer_type   7829 non-null   object
 5   unit_price      7829 non-null   float64
 6   quantity        7829 non-null   int64
 7   total           7829 non-null   float64
 8   payment_type    7829 non-null   object
dtypes: datetime64[ns](1), float64(2), int64(1), object(5)
memory usage: 550.6+ KB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 23890 entries, 0 to 23889
Data columns (total 3 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   id           23890 non-null  object
 1   timestamp    23890 non-null  datetime64[ns]
 2   temperature  23890 non-null  float64
dtypes: datetime64[ns](1), float64(1), object(1)
memory usage: 560.0+ KB
```

In [30]:
```
#We now convert these datetimes to hours
from datetime import datetime

def convert_timestamp_to_hourly(data: pd.DataFrame = None, column: str = None):
    new_ts = data[column].tolist()
    new_ts = map(lambda i:i.strftime('%Y-%m-%d %H:00:00'),new_ts)
    new_ts = [datetime.strptime(i, '%Y-%m-%d %H:00:00') for i in new_ts]
    data[column] = new_ts
```

In [31]:
```
convert_timestamp_to_hourly(sales_data,"timestamp")
convert_timestamp_to_hourly(sensor_storage_temperature,"timestamp")
convert_timestamp_to_hourly(sensor_stock_levels,"timestamp")
```

Now all the timestamp columns have their minutes and seconds reduced to 00. The next thing to do, is to aggregate the datasets in order to combine rows which have the same value for `timestamp`.

For the sales data, we want to group the data by timestamp but also by product_id. When we aggregate, we must choose which columns to aggregate by the grouping. For now, let�s aggregate quantity.

In [32]:
```
sales_agg = sales_data.groupby(['timestamp', 'product_id']).agg({"quantity": 'mean'}).reset_index()
```

```
sales_agg
```

Out[32]:

	timestamp	product_id	quantity
0	2022-03-01 09:00:00	00e120bb-89d6-4df5-bc48-a051148e3d03	3.00
1	2022-03-01 09:00:00	01f3cdd9-8e9e-4dff-9b5c-69698a0388d0	3.00
2	2022-03-01 09:00:00	03a2557a-aa12-4add-a6d4-77dc36342067	3.00
3	2022-03-01 09:00:00	049b2171-0eeb-4a3e-bf98-0c290c7821da	3.50
4	2022-03-01 09:00:00	04da844d-8dba-4470-9119-e534d52a03a0	2.75
...	...	...	...
6212	2022-03-07 19:00:00	edf4ac93-4e14-4a3d-8c60-e715210cf3f9	3.00
6213	2022-03-07 19:00:00	f01b189c-6345-4639-a8d1-89e1fc67c443	3.00
6214	2022-03-07 19:00:00	f3bec808-bee0-4597-a129-53a3a2805a43	2.00
6215	2022-03-07 19:00:00	fd66ac0b-3498-4613-8ec0-764686b0d864	1.00
6216	2022-03-07 19:00:00	fd77b5cb-498c-40ca-95d1-0f87f13dd0d8	1.00

6217 rows × 3 columns

- We now have an aggregated sales data where each row represents a unique combination of hour during which the sales took place from that weeks worth of data and the product_id.
- We summed the quantity and we took the mean average of the unit_price.

For the stock data, we want to group it in the same way and aggregate the estimated_stock_pct.

In [33]:
```
sensor_stock_levels_agg = sensor_stock_levels.groupby(['timestamp', 'product_id']).agg({'estimated_stock_pct':
sensor_stock_levels_agg.head()
```

Out[33]:

	timestamp	product_id	estimated_stock_pct
0	2022-03-01 09:00:00	00e120bb-89d6-4df5-bc48-a051148e3d03	0.89
1	2022-03-01 09:00:00	01f3cdd9-8e9e-4dff-9b5c-69698a0388d0	0.14
2	2022-03-01 09:00:00	01ff0803-ae73-4234-971d-5713c97b7f4b	0.67
3	2022-03-01 09:00:00	0363eb21-8c74-47e1-a216-c37e565e5ceb	0.82
4	2022-03-01 09:00:00	03f0b20e-3b5b-444f-bc39-cdfa2523d4bc	0.05

In [34]:
```
sensor_storage_temperature_agg = sensor_storage_temperature.groupby(['timestamp']).agg({'temperature': 'mean'})
sensor_storage_temperature_agg.head()
```

Out[34]:

	timestamp	temperature
0	2022-03-01 09:00:00	-0.028850
1	2022-03-01 10:00:00	1.284314
2	2022-03-01 11:00:00	-0.560000
3	2022-03-01 12:00:00	-0.537721
4	2022-03-01 13:00:00	-0.188734

This gives us the average temperature of the storage facility where the produce is stored in the warehouse by unique hours during the week. Now, we are ready to merge our data. We will use the `stock_agg` table as our base table, and we will merge our other 2 tables onto this.

In [35]:
```
merged_df = sensor_stock_levels_agg.merge(sales_agg, on=['timestamp', 'product_id'], how='left')
merged_df.head()
```

Out[35]:

	timestamp	product_id	estimated_stock_pct	quantity
0	2022-03-01 09:00:00	00e120bb-89d6-4df5-bc48-a051148e3d03	0.89	3.0
1	2022-03-01 09:00:00	01f3cdd9-8e9e-4dff-9b5c-69698a0388d0	0.14	3.0
2	2022-03-01 09:00:00	01ff0803-ae73-4234-971d-5713c97b7f4b	0.67	NaN
3	2022-03-01 09:00:00	0363eb21-8c74-47e1-a216-c37e565e5ceb	0.82	NaN
4	2022-03-01 09:00:00	03f0b20e-3b5b-444f-bc39-cdfa2523d4bc	0.05	NaN

In [36]:
```
merged_df = merged_df.merge(sensor_storage_temperature_agg, on=['timestamp'], how='left')
merged_df.head()
```

Out[36]:

	timestamp	product_id	estimated_stock_pct	quantity	temperature
0	2022-03-01 09:00:00	00e120bb-89d6-4df5-bc48-a051148e3d03	0.89	3.0	-0.02885
1	2022-03-01 09:00:00	01f3cdd9-8e9e-4dff-9b5c-69698a0388d0	0.14	3.0	-0.02885
2	2022-03-01 09:00:00	01ff0803-ae73-4234-971d-5713c97b7f4b	0.67	NaN	-0.02885
3	2022-03-01 09:00:00	0363eb21-8c74-47e1-a216-c37e565e5ceb	0.82	NaN	-0.02885
4	2022-03-01 09:00:00	03f0b20e-3b5b-444f-bc39-cdfa2523d4bc	0.05	NaN	-0.02885

In [37]:

```
merged_df.isna().sum()
```

Out[37]:

```
timestamp                 0
product_id                0
estimated_stock_pct       0
quantity               7778
temperature               0
dtype: int64
```

We can see from the `.isna` method that we have some null values. These need to be treated before we can build a predictive model. The column that features some null values is `quantity`. We can assume that if there is a null value for this column, it represents that there were 0 sales of this product within this hour. So, lets fill this columns null values with 0, however, we should verify this with the client, in order to make sure we�re not making any assumptions by filling these null values with 0.

In [38]:

```
merged_df['quantity'] = merged_df['quantity'].fillna(0)
merged_df.isna().sum()
```

Out[38]:

```
timestamp              0
product_id             0
estimated_stock_pct    0
quantity               0
temperature            0
dtype: int64
```

We now add these other columns, `category` and `unit_price.`

In [39]:

```
product_categories = sales_data[['product_id', 'category']]
product_categories = product_categories.drop_duplicates()

product_price = sales_data[['product_id', 'unit_price']]
product_price = product_price.drop_duplicates()
product_price
```

Out[39]:

	product_id	unit_price
0	3bc6c1ea-0198-46de-9ffd-514ae3338713	3.99
1	ad81b46c-bf38-41cf-9b54-5fe7f5eba93e	3.99
2	7c55cbd4-f306-4c04-a030-628cbe7867c1	0.19
3	80da8348-1707-403f-8be7-9e6deeccc883	0.19
4	7f5e86e6-f06f-45f6-bf44-27b095c9ad1d	4.49
...	...	...
7569	d6ccd088-11be-4c25-aa1f-ea87c01a04db	14.99
7570	20a9bd7b-daff-4b8b-bdc1-2e8f9a0277fa	13.49
7572	a9325c1a-2715-41df-b7f4-3078fa5ecd97	14.19
7576	0e4c10f4-77bc-4c67-86b2-b4da5ded19bf	16.99
7579	bc6187a9-d508-482b-9ca6-590d1cc7524f	14.19

300 rows × 2 columns

In [40]:

```
merged_df = merged_df.merge(product_categories, on="product_id", how="left")
merged_df = merged_df.merge(product_price, on="product_id", how="left")
merged_df.head()
```

Out[40]:

	timestamp	product_id	estimated_stock_pct	quantity	temperature	category	unit_price
0	2022-03-01 09:00:00	00e120bb-89d6-4df5-bc48-a051148e3d03	0.89	3.0	-0.02885	kitchen	11.19
1	2022-03-01 09:00:00	01f3cdd9-8e9e-4dff-9b5c-69698a0388d0	0.14	3.0	-0.02885	vegetables	1.49
2	2022-03-01 09:00:00	01ff0803-ae73-4234-971d-5713c97b7f4b	0.67	0.0	-0.02885	baby products	14.19
3	2022-03-01 09:00:00	0363eb21-8c74-47e1-a216-c37e565e5ceb	0.82	0.0	-0.02885	beverages	20.19
4	2022-03-01 09:00:00	03f0b20e-3b5b-444f-bc39-cdfa2523d4bc	0.05	0.0	-0.02885	pets	8.19

Section 9 - Feature engineering

We have our cleaned and merged data. Now we must transform this data so that the columns are in a suitable format for a machine learning model. In other terms, every column must be numeric.

Let�s first engineer the `timestamp` column. In it�s current form, it is not very useful for a machine learning model. Since it�s a datetime datatype, we can explode this column into day of week, day of month and hour to name a few.

In [41]:
```
merged_df['timestamp_day_of_month'] = merged_df['timestamp'].dt.day
merged_df['timestamp_day_of_week'] = merged_df['timestamp'].dt.dayofweek
merged_df['timestamp_hour'] = merged_df['timestamp'].dt.hour
merged_df.drop(columns=["product_id","timestamp"], inplace=True)
merged_df.head()
```

Out[41]:

	estimated_stock_pct	quantity	temperature	category	unit_price	timestamp_day_of_month	timestamp_day_of_week	timestamp_hour
0	0.89	3.0	-0.02885	kitchen	11.19	1	1	9
1	0.14	3.0	-0.02885	vegetables	1.49	1	1	9
2	0.67	0.0	-0.02885	baby products	14.19	1	1	9
3	0.82	0.0	-0.02885	beverages	20.19	1	1	9
4	0.05	0.0	-0.02885	pets	8.19	1	1	9

The next column that we can engineer is the `category` column. In its current form it is categorical. We can convert it into numeric by creating dummy variables from this categorical column.

In [42]:
```
merged_df = pd.get_dummies(merged_df,columns=['category'])
merged_df
```

Out[42]:

	estimated_stock_pct	quantity	temperature	unit_price	timestamp_day_of_month	timestamp_day_of_week	timestamp_hour	category_b produ
0	0.89	3.0	-0.028850	11.19	1	1	9	
1	0.14	3.0	-0.028850	1.49	1	1	9	
2	0.67	0.0	-0.028850	14.19	1	1	9	
3	0.82	0.0	-0.028850	20.19	1	1	9	
4	0.05	0.0	-0.028850	8.19	1	1	9	
...	...	...	...	...	...	...	...	
10840	0.50	4.0	-0.165077	4.99	7	0	19	
10841	0.26	0.0	-0.165077	19.99	7	0	19	
10842	0.78	3.0	-0.165077	6.99	7	0	19	
10843	0.92	3.0	-0.165077	14.99	7	0	19	
10844	0.01	2.0	-0.165077	5.19	7	0	19	

10845 rows × 29 columns

Section 10 - Modelling

Now it is time to train a machine learning model. We used a supervised machine learning model using `estimated_stock_pct` as the target variable, since the problem statement was focused on being able to predict the stock levels of products on an hourly basis.

Whilst training the machine learning model, we will use cross-validation, which is a technique where we hold back a portion of the dataset for testing in order to compute how well the trained machine learning model is able to predict the target variable.

Finally, to ensure that the trained machine learning model is able to perform robustly, we will want to test it several times on random samples of data, not just once. Hence, we will use a K-fold strategy to train the machine learning model on K (K is an integer to be decided) random samples of the data. First, let�s create our target variable y and independent variables X

In [43]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error
from sklearn.preprocessing import StandardScaler
```

In [44]:
```
features = merged_df.drop(columns=['estimated_stock_pct'])
target = merged_df['estimated_stock_pct']
print(features.shape)
print(target.shape)
```

```
(10845, 28)
(10845,)
```

For this exercise, we are going to use a RandomForestRegressor model, which is an instance of a Random Forest. These are powerful

For this exercise, we are going to use a RandomForestRegressor model, which is an instance of a Random Forest. These are powerful tree based ensemble algorithms and are particularly good because their results are very interpretable.

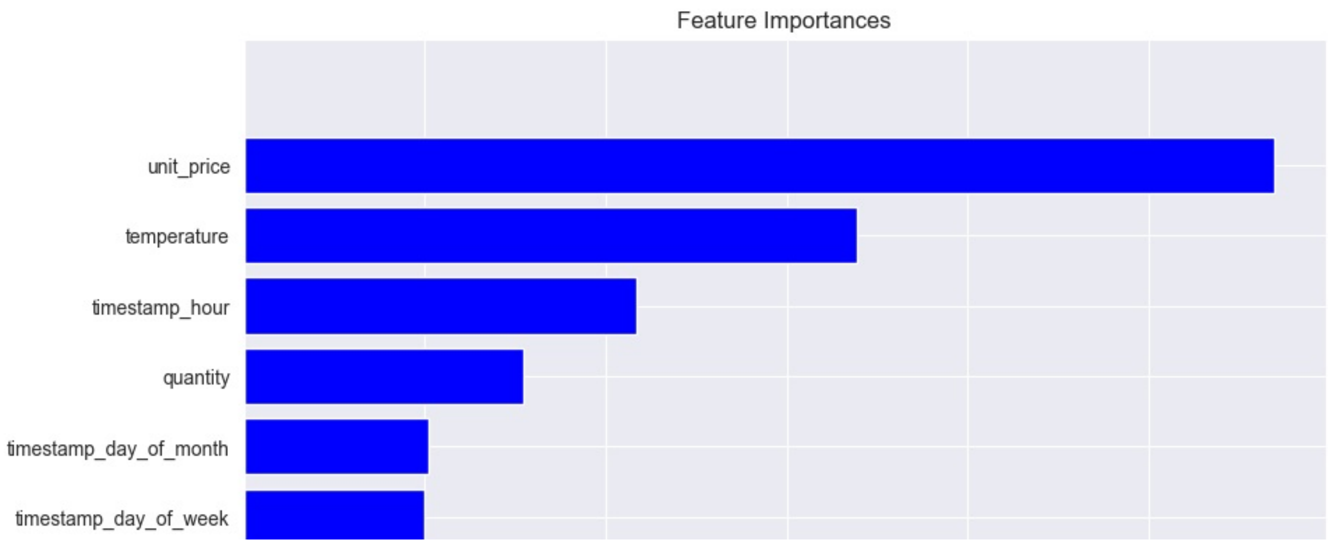
We are using a regression algorithm here because we are predicting a continuous numeric variable, that is, estimated_stock_pct. A classification algorithm would be suitable for scenarios where you�re predicted a binary outcome, e.g. True/False

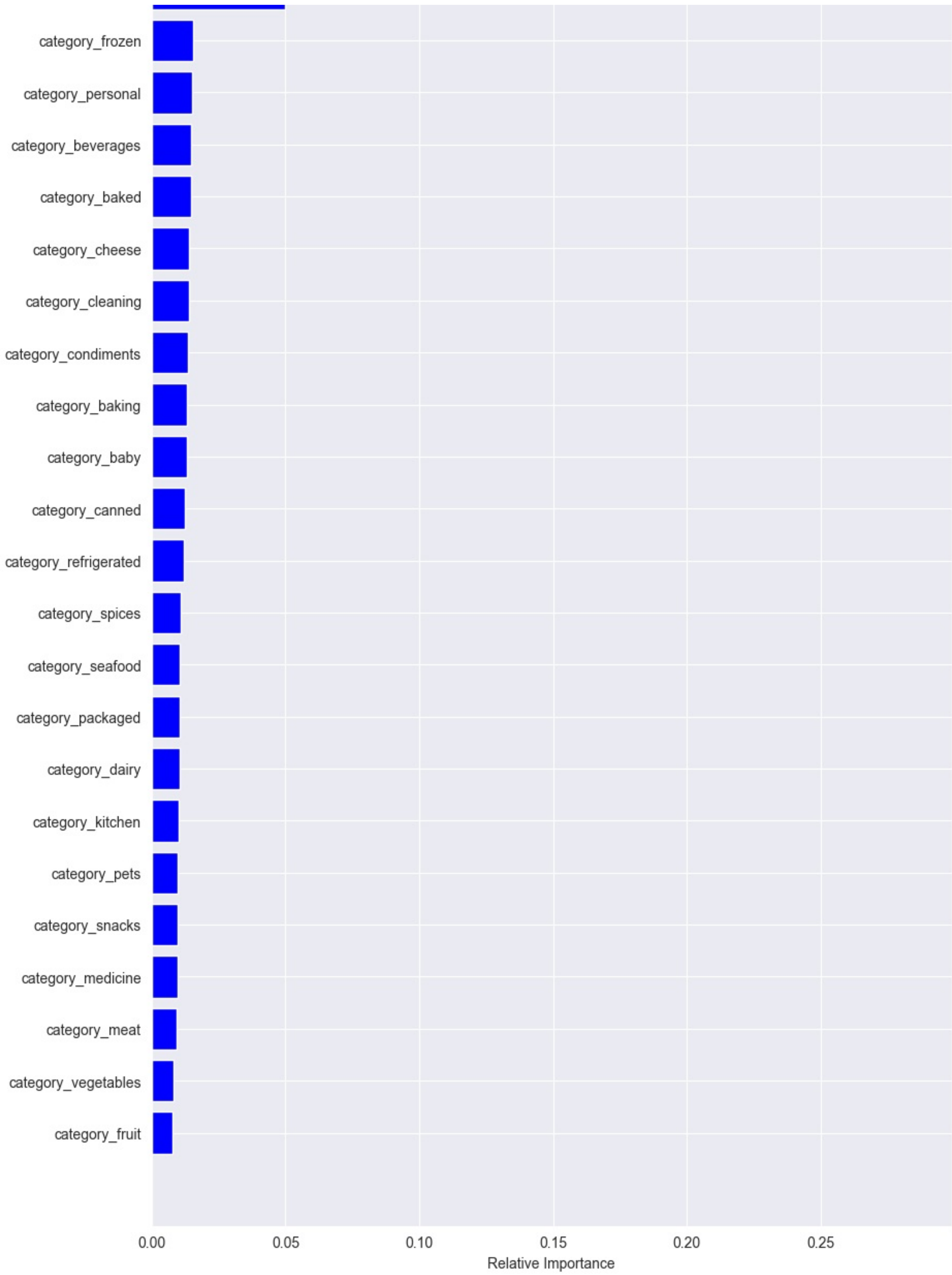
In [45]:
```
K = 10
split = 0.75
```

In [46]:
```
accuracy = []

for fold in range(0, K):

    # Instantiate algorithm
    model = RandomForestRegressor()
    scaler = StandardScaler()

    # Create training and test samples
    features_train, features_test, target_train, target_test = train_test_split(features, target, train_size=sp

    # Scale X data, we scale the data because it helps the algorithm to converge
    # and helps the algorithm to not be greedy with large values
    scaler.fit(features_train)
    X_train = scaler.transform(features_train)
    X_test = scaler.transform(features_test)

    # Train model
    trained_model = model.fit(features_train, target_train)

    # Generate predictions on test sample
    y_pred = trained_model.predict(features_test)

    # Compute accuracy, using mean absolute error
    mae = mean_absolute_error(target_test,y_pred)
    accuracy.append(mae)
    print(f"Fold {fold + 1}: MAE = {mae:.3f}")

print(f"Average MAE: {(sum(accuracy) / len(accuracy)):.2f}")
```

```
Fold 1: MAE = 0.237
Fold 2: MAE = 0.236
Fold 3: MAE = 0.236
Fold 4: MAE = 0.236
Fold 5: MAE = 0.236
Fold 6: MAE = 0.236
Fold 7: MAE = 0.236
Fold 8: MAE = 0.237
Fold 9: MAE = 0.237
Fold 10: MAE = 0.236
Average MAE: 0.24
```

In [47]:
```
features = [i.split()[0] for i in features.columns]
importances = model.feature_importances_

indices = np.argsort(importances)

fig, ax = plt.subplots(figsize=(10, 20))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='b', align='center')
plt.yticks(range(len(indices)), [features[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```





This feature importance visualisation tells us:

1. The product categories were not that important
2. The unit price and temperature were important in predicting stock
3. The hour of day was also important for predicting stock