PROBLEM STATEMENT DESCRIPTION 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.

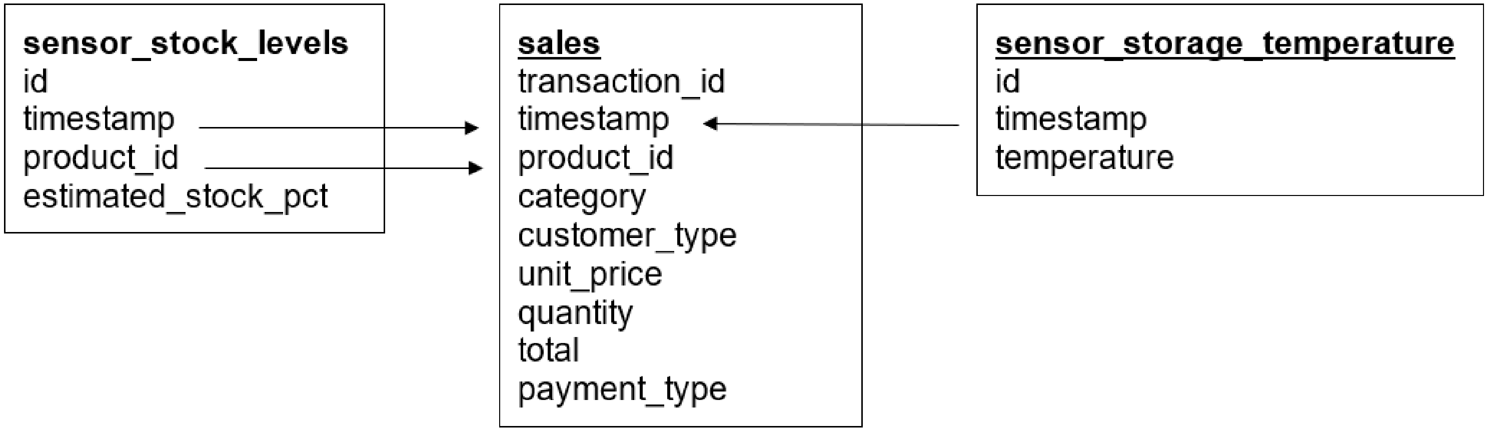
They pride themselves on providing the best quality, fresh produce from locally sourced suppliers. However, this comes with many challenges to consistently deliver on this objective year-round. Groceries are highly perishable items. If you overstock, you are wasting money on excessive storage and waste, but if you understock, then you risk losing customers. They want to know how to better stock the items that they sell.

TASK1 Perform an initial level EDA with sample dataset made availbale

“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 has agreed to share more data in the form of sensor data. They use sensors to measure temperature storage facilities where products are stored in the warehouse, and they also use stock levels within the refrigerators and freezers in store. consider this diagram given below:

Step 1: Data modeling Look at the data model provided in the additional resources. Look at all the data that is now available from the client and decide what you want to use for the modeling of the problem statement. Step 2: Strategic planning Come up with a plan as to how you’ll use this data to solve the problem statement that the client has positioned. This plan will be used to describe to the client how we are planning to complete the remaining work and to build trust with the client as a domain expert. If you need some guidance, use the provided resource video that describes the high-level overview of a data science project STEP3: The client has provided 3 datasets, it is now your job to combine, transform and model these datasets in a suitable way to answer the problem statement that the business has requested. Remember 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?”



This data model diagram shows:

- 3 tables:
  - sales = sales data
  - sensor_storage_temperature = IoT data from the temperature sensors in the storage facility for the products
  - sensor_stock_levels = estimated stock levels of products based on IoT sensors
- Relations between tables
  - These are shown by the arrows. Make note of the columns that connect the start and end of the arrows, this indicates how you can merge the tables using these linked columns.

In [1]:
```
import pandas as pd
```

In [2]:
```
path = "C:/Users/ASUS/Downloads/cognizant forage/task1/sample_sales_data.csv"
df = pd.read_csv(path)
df.drop(columns=["Unnamed: 0"], inplace=True, errors='ignore')
df.head()
```

Out[2]:

	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

Descriptive statistics

Descriptive statistics In this section, you should 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 you started an explanation of what the column names mean are provided below:

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 After this, you should try to compute some descriptive statistics of the numerical columns within the dataset, such as:

mean median count etc...

In [3]:

```
df.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 [4]:
```
df.describe()
```

Out[4]:

	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

In [5]:
```
df.isna().sum()/len(df)*100
```

Out[5]:
```
transaction_id    0.0
timestamp         0.0
product_id        0.0
category          0.0
customer_type     0.0
unit_price        0.0
quantity          0.0
total             0.0
payment_type      0.0
dtype: float64
```

In [6]:
```
df["unit_price"].mean()
```

Out[6]:
```
7.819480137948519
```

In [7]:
```
df["unit_price"].median()
```

Out[7]:
```
7.19
```

In [8]:
```
df["unit_price"].count()
```

Out[8]:
```
7829
```

In [9]:
```
df["total"].mean()
```

Out[9]:
```
19.70990547962791
```

In [10]:
```
df["total"].median()
```

Out[10]:
```
14.97
```

In [11]:
```
df["total"].count()
```

Out[11]:
```
7829
```

In [12]:
```
df["quantity"].mean()
```

Out[12]:
```
2.501596627921829
```

In [13]:
```
df["quantity"].median()
```

Out[13]: 3.0

In [14]:
```
df["quantity"].count()
```

Out[14]: 7829

Visualisation

Now that you've computed some descriptive statistics of the dataset, let's create some visualisations. You may use any package that you wish for visualisation, however, some helper functions have been provided that make use of the seaborn package. If you wish to use these helper functions, ensure to run the below cells that install and import seaborn.

In [15]:
```
!pip install seaborn
```

```
Requirement already satisfied: seaborn in c:\users\asus\anaconda3\lib\site-packages (0.11.2)
Requirement already satisfied: pandas>=0.23 in c:\users\asus\anaconda3\lib\site-packages (from seaborn) (1.4.4)
Requirement already satisfied: scipy>=1.0 in c:\users\asus\anaconda3\lib\site-packages (from seaborn) (1.9.1)
Requirement already satisfied: matplotlib>=2.2 in c:\users\asus\anaconda3\lib\site-packages (from seaborn) (3.5.2)
Requirement already satisfied: numpy>=1.15 in c:\users\asus\anaconda3\lib\site-packages (from seaborn) (1.21.5)
Requirement already satisfied: pyparsing>=2.2.1 in c:\users\asus\anaconda3\lib\site-packages (from matplotlib>=2.2->sea
born) (3.0.9)
Requirement already satisfied: packaging>=20.0 in c:\users\asus\anaconda3\lib\site-packages (from matplotlib>=2.2->seab
orn) (21.3)
Requirement already satisfied: cycler>=0.10 in c:\users\asus\anaconda3\lib\site-packages (from matplotlib>=2.2->seabor
n) (0.11.0)
Requirement already satisfied: pillow>=6.2.0 in c:\users\asus\anaconda3\lib\site-packages (from matplotlib>=2.2->seabor
n) (9.2.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\asus\anaconda3\lib\site-packages (from matplotlib>=2.2->se
aborn) (1.4.2)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\asus\anaconda3\lib\site-packages (from matplotlib>=2.2-
>seaborn) (2.8.2)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\asus\anaconda3\lib\site-packages (from matplotlib>=2.2->se
aborn) (4.25.0)
Requirement already satisfied: pytz>=2020.1 in c:\users\asus\anaconda3\lib\site-packages (from pandas>=0.23->seaborn)
(2022.1)
Requirement already satisfied: six>=1.5 in c:\users\asus\anaconda3\lib\site-packages (from python-dateutil>=2.7->matplo
tlib>=2.2->seaborn) (1.16.0)
```

In [16]:
```
import seaborn as sns
```

To analyse the dataset, below are snippets of code that you can use as helper functions to visualise different columns within the dataset. They include:

plot_continuous_distribution = this is to visualise the distribution of numeric columns get_unique_values = this is to show how many unique values are present within a column plot_categorical_distribution = this is to visualise the distribution of categorical columns correlation_plot = this is to plot the correlations between the numeric columns within the data

In [17]:

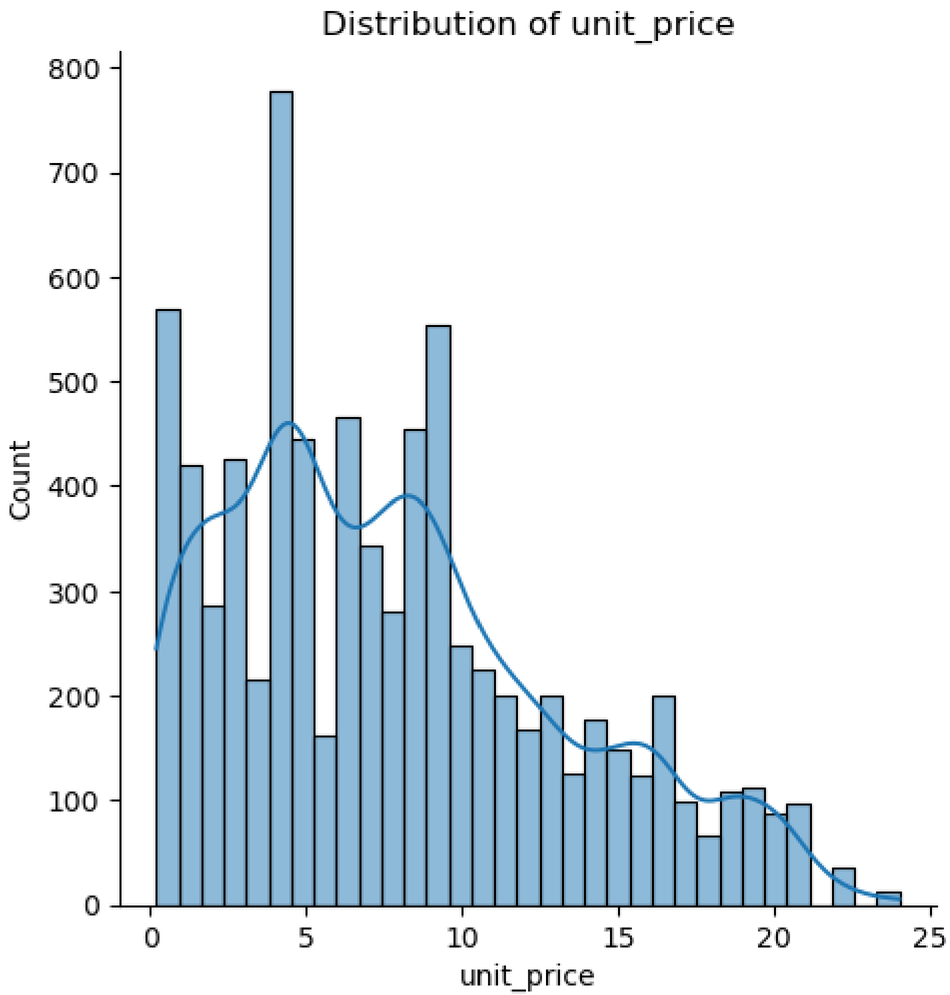
```
def plot_continuous_distribution(data: pd.DataFrame = None, column: str = None, height: int = 8):
  _ = sns.displot(data, x=column, kde=True, height=height, aspect=height/5).set(title=f'Distribution of {column}');

def get_unique_values(data, column):
  num_unique_values = len(data[column].unique())
  value_counts = data[column].value_counts()
  print(f"Column: {column} has {num_unique_values} unique values\n")
  print(value_counts)

def plot_categorical_distribution(data: pd.DataFrame = None, column: str = None, height: int = 8, aspect: int = 2):
  _ = sns.catplot(data=data, x=column, kind='count', height=height, aspect=aspect).set(title=f'Distribution of {column}

def correlation_plot(data: pd.DataFrame = None):
  corr = df.corr()
  corr.style.background_gradient(cmap='coolwarm')
```

In [18]:

```
sns.displot(data=df, x="unit_price", kde=True, ).set(title=f'Distribution of {"unit_price"}')
```

Out[18]:

```
<seaborn.axisgrid.FacetGrid at 0x233d9b3f280>
```



In [19]:
```
get_unique_values(df, "unit_price")
```

```
Column: unit_price has 64 unique values

3.99     374
4.99     374
1.49     321
0.49     306
8.19     272
        ...
21.99     17
20.99     17
23.99     13
17.99     12
20.19     11
Name: unit_price, Length: 64, dtype: int64
```

In [20]:
```
get_unique_values(df, "total")
```

```
Column: total has 256 unique values

14.97    104
3.99     103
11.97     98
4.99      94
19.96     94
        ...
60.57      2
47.98      2
17.99      2
20.19      1
35.98      1
Name: total, Length: 256, dtype: int64
```

In [21]:
```
get_unique_values(df, "quantity")
```

```
Column: quantity has 4 unique values

1    1979
4    1976
3    1954
2    1920
Name: quantity, dtype: int64
```

In [22]:
```
get_unique_values(df, "category")
```

```
Column: category has 22 unique values

fruit                    998
vegetables               846
packaged foods           507
baked goods              443
canned foods             431
refrigerated items       425
kitchen                  382
meat                     382
dairy                    375
beverages                301
cheese                   293
cleaning products        292
baking                   264
snacks                   263
frozen                   263
seafood                  253
medicine                 243
baby products            224
condiments and sauces    181
personal care            177
pets                     161
spices and herbs         125
Name: category, dtype: int64
```

In [23]:
```python
get_unique_values(df, "customer_type")
```

```
Column: customer_type has 5 unique values

non-member    1601
standard      1595
premium       1590
basic         1526
gold          1517
Name: customer_type, dtype: int64
```

In [24]:
```python
get_unique_values(df, "payment_type")
```

```
Column: payment_type has 4 unique values

cash           2027
credit card    1949
e-wallet       1935
debit card     1918
Name: payment_type, dtype: int64
```

In [25]:
```python
get_unique_values(df, "unit_price")
```

```
Column: unit_price has 64 unique values

3.99     374
4.99     374
1.49     321
0.49     306
8.19     272
        ... 
21.99     17
20.99     17
23.99     13
17.99     12
20.19     11
Name: unit_price, Length: 64, dtype: int64
```

In [26]:
```python
get_unique_values(df, "product_id")
```

```
Column: product_id has 300 unique values

ecac012c-1dec-41d4-9ebd-56fb7166f6d9    114
80da8348-1707-403f-8be7-9e6deeccc883    109
0ddc2379-adba-4fb0-aa97-19fcafc738a1    108
7c55cbd4-f306-4c04-a030-628cbe7867c1    104
3bc6c1ea-0198-46de-9ffd-514ae3338713    101
                                       ... 
49f7d4a9-713a-4824-b378-aebb33ff8b2f      5
a8fab83a-16d4-4db0-a83a-f824ecd8604a      5
c8de27d0-2c44-4b5a-b178-59c45d054ccb      5
5adfc643-aa8e-4140-b2c3-98a946444632      5
ec0bb9b5-45e3-4de8-963d-e92aa91a201e      3
Name: product_id, Length: 300, dtype: int64
```

In [27]:
```python
get_unique_values(df, "transaction_id")
```

```
Column: transaction_id has 7829 unique values

a1c82654-c52c-45b3-8ce8-4c2a1efe63ed    1
6532e258-95fd-4eb5-8c67-2bfb879a8fec    1
6fce2af3-47a0-4755-99c9-0cefb5ab6f41    1
6476e388-3990-471f-b415-3ee59ae18832    1
10afe89b-c45b-49a2-b0be-dec89a4c3f80    1
                                       ..
a9abe5ac-99d5-4d8b-bbbd-c2a207642849    1
6b0b23e8-412b-4665-8cc4-3e37f0d9e195    1
711a4162-1985-4f5a-94ca-137cfacaeadf    1
7d1e9010-dbaf-4770-a467-f31477910f7a    1
afd70b4f-ee21-402d-8d8f-0d9e13c2bea6    1
Name: transaction_id, Length: 7829, dtype: int64
```

In [28]:

```python
sns.catplot(data=df, x="category", kind='count', height=10, aspect=10/5).set(title=f'Distribution of {"category"}')
```

Out[28]: <seaborn.axisgrid.FacetGrid at 0x233d973d790>

In [29]:

```python
sns.catplot(data=df, x="customer_type", kind='count', height=8, aspect=8/5).set(title=f'Distribution of {"customer_type
```

Out[29]: <seaborn.axisgrid.FacetGrid at 0x233dd4f1cd0>

In [30]:

```python
sns.catplot(data=df, x="payment_type", kind='count', height=8, aspect=8/5).set(title=f'Distribution of {"payment_type"}
```

Out[30]: <seaborn.axisgrid.FacetGrid at 0x233dd4f7100>

In [31]:

```python
import  numpy as np
numerical_df = df.select_dtypes(include=[np.number])
```

In [32]:

```python
numerical_df.corr()
```

Out[32]:

	unit_price	quantity	total
unit_price	1.000000	0.024588	0.792018
quantity	0.024588	1.000000	0.521926
total	0.792018	0.521926	1.000000

Summary of EDA

We have completed an initial exploratory data analysis on the sample of data provided. We should now have a solid understanding of the data.

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

In [33]:
```
import pandas as pd
import numpy as np
```

We want to use dataframes once again to store and manipulate the data.

Section 2 - Data loading

Similar to before, let's load our data from Google Drive for the 3 datasets provided. Be sure to upload the datasets into Google Drive, so that you can access them here.

In [34]:
```
sales_df = pd.read_csv("sales.csv")
```

In [35]:
```
sales_df.drop(columns=["Unnamed: 0"], inplace=True, errors='ignore')
sales_df.head()
```

Out[35]:

	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

In [36]:

```
stock_df = pd.read_csv("sensor_stock_levels.csv")
stock_df.drop(columns=["Unnamed: 0"], inplace=True, errors='ignore')
stock_df.head()
```

Out[36]:

	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

In [37]:

```
temp_df = pd.read_csv("sensor_storage_temperature.csv")
temp_df.drop(columns=["Unnamed: 0"], inplace=True, errors='ignore')
temp_df.head()
```

Out[37]:

	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

Section 3 - Data cleaning

data Cleaning involves alot of steps checking for null values It also includes checking the data types of each of the columns for the different datasets

In [38]:
```
sales_df.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 [39]:
```
stock_df.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 [40]:
```
temp_df.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 [41]:
```
temp_df.isna().sum()/len(temp_df)*100
```

Out[41]:
```
id             0.0
timestamp      0.0
temperature    0.0
dtype: float64
```

In [42]:
```
stock_df.isna().sum()/len(stock_df)*100
```

Out[42]:
```
id                     0.0
timestamp              0.0
product_id             0.0
estimated_stock_pct    0.0
dtype: float64
```

In [43]:
```
sales_df.isna().sum()/len(stock_df)*100
```

Out[43]:
```
transaction_id    0.0
timestamp         0.0
product_id        0.0
category          0.0
customer_type     0.0
unit_price        0.0
quantity          0.0
total             0.0
payment_type      0.0
dtype: float64
```

The three datasets have no null values , however to further merge the three datasets it is important to consider the column timesatmp which has to changed to correct data type

In [44]:
```
sales_df['timestamp'] = pd.to_datetime(sales_df['timestamp'], format='%Y-%m-%d %H:%M:%S')
```

In [45]:
```
sales_df.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   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
```

In [46]:
```
stock_df["timestamp"]=pd.to_datetime(stock_df['timestamp'], format='%Y-%m-%d %H:%M:%S')
```

In [47]:
```
stock_df.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
```

In [48]:
```
temp_df["timestamp"]=pd.to_datetime(temp_df['timestamp'],format='%Y-%m-%d %H:%M:%S')
```

In [49]:
```
temp_df.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  datetime64[ns]
 2   temperature  23890 non-null  float64
dtypes: datetime64[ns](1), float64(1), object(1)
memory usage: 560.0+ KB
```

we have three datasets our problem statemet is to "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.

So, we must first transform the timestamp column in all 3 datasets to be based on the hour of the day, then we can merge the datasets together.

In [50]:
```
sales_df['timestamp'] = sales_df['timestamp'].dt.strftime('%Y-%m-%d %H:00:00')
```

In [51]:
```
sales_df.head(5)
```

Out[51]:

	transaction_id	timestamp	product_id	category	customer_type	unit_price	quantity	total	payment_type
0	a1c82654-c52c-45b3-8ce8-4c2a1efe63ed	2022-03-02 09:00:00	3bc6c1ea-0198-46de-9ffd-514ae3338713	fruit	gold	3.99	2	7.98	e-wallet
1	931ad550-09e8-4da6-beaa-8c9d17be9c60	2022-03-06 10:00:00	ad81b46c-bf38-41cf-9b54-5fe7f5eba93e	fruit	standard	3.99	1	3.99	e-wallet
2	ae133534-6f61-4cd6-b6b8-d1c1d8d90aea	2022-03-04 17:00:00	7c55cbd4-f306-4c04-a030-628cbe7867c1	fruit	premium	0.19	2	0.38	e-wallet
3	157cebd9-aaf0-475d-8a11-7c8e0f5b76e4	2022-03-02 17:00:00	80da8348-1707-403f-8be7-9e6deeccc883	fruit	gold	0.19	4	0.76	e-wallet
4	a81a6cd3-5e0c-44a2-826c-aea43e46c514	2022-03-05 14:00:00	7f5e86e6-f06f-45f6-bf44-27b095c9ad1d	fruit	basic	4.49	2	8.98	debit card

In [52]:
```
stock_df["timestamp"]= stock_df["timestamp"].dt.strftime('%Y-%m-%d %H:00:00')
```

In [53]:
```
stock_df.head()
```

Out[53]:

	id	timestamp	product_id	estimated_stock_pct
0	4220e505-c247-478d-9831-6b9f87a4488a	2022-03-07 12:00:00	f658605e-75f3-4fed-a655-c0903f344427	0.75
1	f2612b26-fc82-49ea-8940-0751fdd4d9ef	2022-03-07 16:00:00	de06083a-f5c0-451d-b2f4-9ab88b52609d	0.48
2	989a287f-67e6-4478-aa49-c3a35dac0e2e	2022-03-01 18:00:00	ce8f3a04-d1a4-43b1-a7c2-fa1b8e7674c8	0.58
3	af8e5683-d247-46ac-9909-1a77bdebefb2	2022-03-02 14:00:00	c21e3ba9-92a3-4745-92c2-6faef73223f7	0.79
4	08a32247-3f44-4002-85fb-c198434dd4bb	2022-03-02 13:00:00	7f478817-aa5b-44e9-9059-8045228c9eb0	0.22

In [54]:
```
temp_df["timestamp"]= temp_df["timestamp"].dt.strftime('%Y-%m-%d %H:00:00')
```

In [55]:
```
temp_df.head()
```

Out[55]:

	id	timestamp	temperature
0	d1ca1ef8-0eac-42fc-af80-97106efc7b13	2022-03-07 15:00:00	2.96
1	4b8a66c4-0f3a-4f16-826f-8cf9397e9d18	2022-03-01 09:00:00	1.88
2	3d47a0c7-1e72-4512-812f-b6b5d8428cf3	2022-03-04 15:00:00	1.78
3	9500357b-ce15-424a-837a-7677b386f471	2022-03-02 12:00:00	2.18
4	c4b61fec-99c2-4c6d-8e5d-4edd8c9632fa	2022-03-05 09:00:00	1.38

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 [56]:
```
sales_agg = sales_df.groupby(['timestamp', 'product_id']).agg({'quantity': 'sum'}).reset_index()
sales_agg.head()
```

Out[56]:

	timestamp	product_id	quantity
0	2022-03-01 09:00:00	00e120bb-89d6-4df5-bc48-a051148e3d03	3
1	2022-03-01 09:00:00	01f3cdd9-8e9e-4dff-9b5c-69698a0388d0	3
2	2022-03-01 09:00:00	03a2557a-aa12-4add-a6d4-77dc36342067	3
3	2022-03-01 09:00:00	049b2171-0eeb-4a3e-bf98-0c290c7821da	7
4	2022-03-01 09:00:00	04da844d-8dba-4470-9119-e534d52a03a0	11

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 [57]:
```
stock_agg = stock_df.groupby(['timestamp', 'product_id']).agg({'estimated_stock_pct': 'mean'}).reset_index()
stock_agg.head()
```

Out[57]:

	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

This shows us the average stock percentage of each product at unique hours within the week of sample data.

Finally, for the temperature data, product_id does not exist in this table, so we simply need to group by timestamp and aggregate the temperature.

In [58]:
```
temp_agg = temp_df.groupby(['timestamp']).agg({'temperature': 'mean'}).reset_index()
temp_agg.head()
```

Out[58]:

	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 [59]:
```
merged_df=stock_agg.merge(sales_agg,on =["timestamp","product_id"],how='left')
merged_df.head()
```

Out[59]:

	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 [60]:

```
merged_df = merged_df.merge(temp_agg, on='timestamp', how='left')
merged_df.head()
```

Out[60]:

	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 [61]:

```
merged_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10845 entries, 0 to 10844
Data columns (total 5 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   timestamp            10845 non-null  object 
 1   product_id           10845 non-null  object 
 2   estimated_stock_pct  10845 non-null  float64
 3   quantity             3067 non-null   float64
 4   temperature          10845 non-null  float64
dtypes: float64(3), object(2)
memory usage: 508.4+ KB
```

We can see from the .info() 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 [62]:
```
merged_df['quantity'] = merged_df['quantity'].fillna(0)
merged_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10845 entries, 0 to 10844
Data columns (total 5 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   timestamp            10845 non-null  object
 1   product_id           10845 non-null  object
 2   estimated_stock_pct  10845 non-null  float64
 3   quantity             10845 non-null  float64
 4   temperature          10845 non-null  float64
dtypes: float64(3), object(2)
memory usage: 508.4+ KB
```

We can combine some more features onto this table too, including category and unit_price.

In [63]:
```
product_categories = sales_df[['product_id', 'category']]
product_categories = product_categories.drop_duplicates()

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

In [64]:
```
merged_df = merged_df.merge(product_categories, on="product_id", how="left")
merged_df.head()
```

Out[64]:

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

In [65]:
```
merged_df = merged_df.merge(product_price, on="product_id", how="left")
merged_df.head()
```

Out[65]:

	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

In [66]:

```
merged_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10845 entries, 0 to 10844
Data columns (total 7 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   timestamp            10845 non-null  object 
 1   product_id           10845 non-null  object 
 2   estimated_stock_pct  10845 non-null  float64
 3   quantity             10845 non-null  float64
 4   temperature          10845 non-null  float64
 5   category             10845 non-null  object 
 6   unit_price           10845 non-null  float64
dtypes: float64(4), object(3)
memory usage: 677.8+ KB
```

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. There are some models that will accept categorical features, but for this exercise we will use a model that requires numeric features.

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 [67]:

```
from datetime import datetime
merged_df['timestamp'] = pd.to_datetime(merged_df['timestamp'], format='%Y-%m-%d %H:%M:%S')
```

In [68]:
```
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=['timestamp'], inplace=True)
merged_df.head()
```

Out[68]:

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

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.

A dummy variable is a binary flag column (1's and 0's) that indicates whether a row fits a particular value of that column. For example, we can create a dummy column called category_pets, which will contain a 1 if that row indicates a product which was included within this category and a 0 if not.

In [69]:
```
merged_df = pd.get_dummies(merged_df, columns=['category'])
merged_df.head()
```

Out[69]:

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

5 rows × 30 columns

In [70]:

```
merged_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10845 entries, 0 to 10844
Data columns (total 30 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   product_id                      10845 non-null  object 
 1   estimated_stock_pct             10845 non-null  float64
 2   quantity                        10845 non-null  float64
 3   temperature                     10845 non-null  float64
 4   unit_price                      10845 non-null  float64
 5   timestamp_day_of_month          10845 non-null  int64  
 6   timestamp_day_of_week           10845 non-null  int64  
 7   timestamp_hour                  10845 non-null  int64  
 8   category_baby products          10845 non-null  uint8  
 9   category_baked goods            10845 non-null  uint8  
 10  category_baking                 10845 non-null  uint8  
 11  category_beverages              10845 non-null  uint8  
 12  category_canned foods           10845 non-null  uint8  
 13  category_cheese                 10845 non-null  uint8  
 14  category_cleaning products      10845 non-null  uint8  
 15  category_condiments and sauces  10845 non-null  uint8  
 16  category_dairy                  10845 non-null  uint8  
 17  category_frozen                 10845 non-null  uint8  
 18  category_fruit                  10845 non-null  uint8  
 19  category_kitchen                10845 non-null  uint8  
 20  category_meat                   10845 non-null  uint8  
 21  category_medicine               10845 non-null  uint8  
 22  category_packaged foods         10845 non-null  uint8  
 23  category_personal care          10845 non-null  uint8  
 24  category_pets                   10845 non-null  uint8  
 25  category_refrigerated items     10845 non-null  uint8  
 26  category_seafood                10845 non-null  uint8  
 27  category_snacks                 10845 non-null  uint8  
 28  category_spices and herbs       10845 non-null  uint8  
 29  category_vegetables             10845 non-null  uint8  
dtypes: float64(4), int64(3), object(1), uint8(22)
memory usage: 995.5+ KB
```

Looking at the latest table, we only have 1 remaining column which is not numeric. This is the product_id.

Since each row represents a unique combination of product_id and timestamp by hour, and the product_id is simply an ID column, it will add no value by including it in the predictive model. Hence, we shall remove it from the modeling process.

In [71]:
```
merged_df.drop(columns=['product_id'], inplace=True)
merged_df.head()
```

Out[71]:

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

5 rows × 29 columns

This feature engineering was by no means exhaustive, but was enough to give you an example of the process followed when engineering the features of a dataset. In reality, this is an iterative task. Once you've built a model, you may have to revist feature engineering in order to create new features to boost the predictive power of a machine learning model.

Modelling Now it is time to train a machine learning model. We will use a supervised machine learning model, and we will use 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 [72]:
```
X = merged_df.drop(columns=['estimated_stock_pct'])
y = merged_df['estimated_stock_pct']
print(X.shape)
print(y.shape)
```

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

This shows that we have 29 predictor variables that we will train our machine learning model on and 10845 rows of data.

Now let's define how many folds we want to complete during training, and how much of the dataset to assign to training, leaving the rest for test.

Typically, we should leave at least 20-30% of the data for testing.

In [73]:
```
K = 10
split = 0.75
```

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.

We are going to use a package called scikit-learn for the machine learning algorithm, so first we must install and import this, along with some other functions and classes that can help with the evaluation of the model.

In [74]:
```
!pip install scikit-learn
```

```
Requirement already satisfied: scikit-learn in c:\users\asus\anaconda3\lib\site-packages (1.0.2)
Requirement already satisfied: joblib>=0.11 in c:\users\asus\anaconda3\lib\site-packages (from scikit-learn) (1.1.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\asus\anaconda3\lib\site-packages (from scikit-learn)
(2.2.0)
Requirement already satisfied: scipy>=1.1.0 in c:\users\asus\anaconda3\lib\site-packages (from scikit-learn) (1.9.1)
Requirement already satisfied: numpy>=1.14.6 in c:\users\asus\anaconda3\lib\site-packages (from scikit-learn) (1.21.5)
```

In [75]:
```
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
```

And now let's create a loop to train K models with a 75/25% random split of the data each time between training and test samples

In [76]:
```
accuracy = []
```

```python
for fold in range(0, K):

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

  # Create training and test samples
  X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=split, random_state=42)

  # 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(X_train)
  X_train = scaler.transform(X_train)
  X_test = scaler.transform(X_test)

  # Train model
  trained_model = model.fit(X_train, y_train)

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

  # Compute accuracy, using mean absolute error
  mae = mean_absolute_error(y_true=y_test, y_pred=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 = 0.237
Fold 3: MAE = 0.237
Fold 4: MAE = 0.236
Fold 5: MAE = 0.237
Fold 6: MAE = 0.237
Fold 7: MAE = 0.236
Fold 8: MAE = 0.236
Fold 9: MAE = 0.236
Fold 10: MAE = 0.236
Average MAE: 0.24
```

Note, the output of this training loop may be slightly different for you if you have prepared the data differently or used different parameters!

This is very interesting though. We can see that the mean absolute error (MAE) is almost exactly the same each time. This is a good sign, it shows that the performance of the model is consistent across different random samples of the data, which is what we want. In other words, it shows a robust nature.

The MAE was chosen as a performance metric because it describes how closely the machine learning model was able to predict the exact value of estimated_stock_pct.

Even though the model is predicting robustly, this value for MAE is not so good, since the average value of the target variable is around 0.51, meaning that the accuracy as a percentage was around 50%. In an ideal world, we would want the MAE to be as low as possible. This is where the iterative process of machine learning comes in. At this stage, since we only have small samples of the data, we can report back to the business with these findings and recommend that the the dataset needs to be further engineered, or more datasets need to be added.

As a final note, we can use the trained model to intepret which features were signficant when the model was predicting the target variable. We will use matplotlib and numpy to visualuse the results, so we should install and import this package.

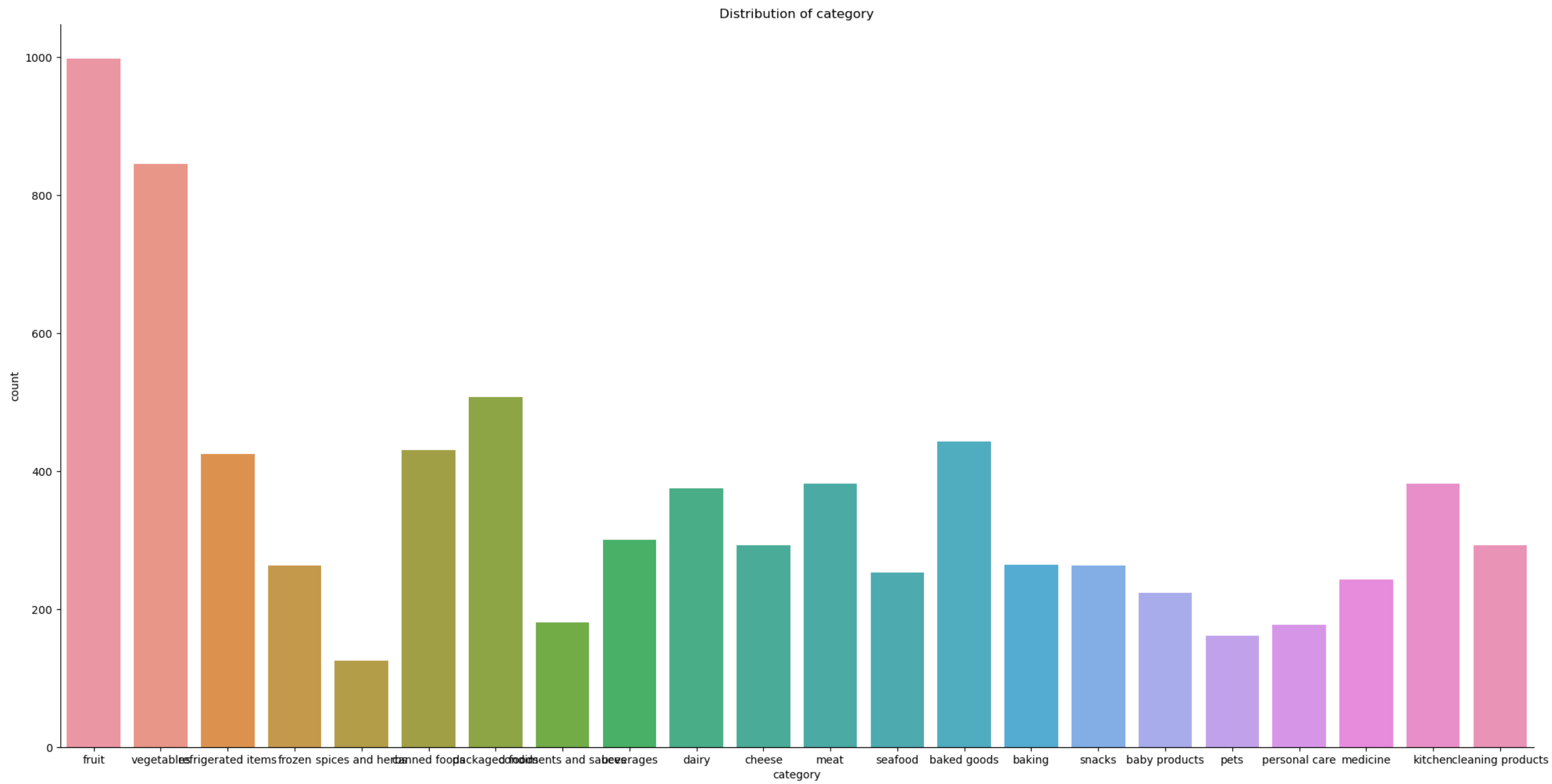
In [77]:
```
!pip install matplotlib
!pip install numpy
```

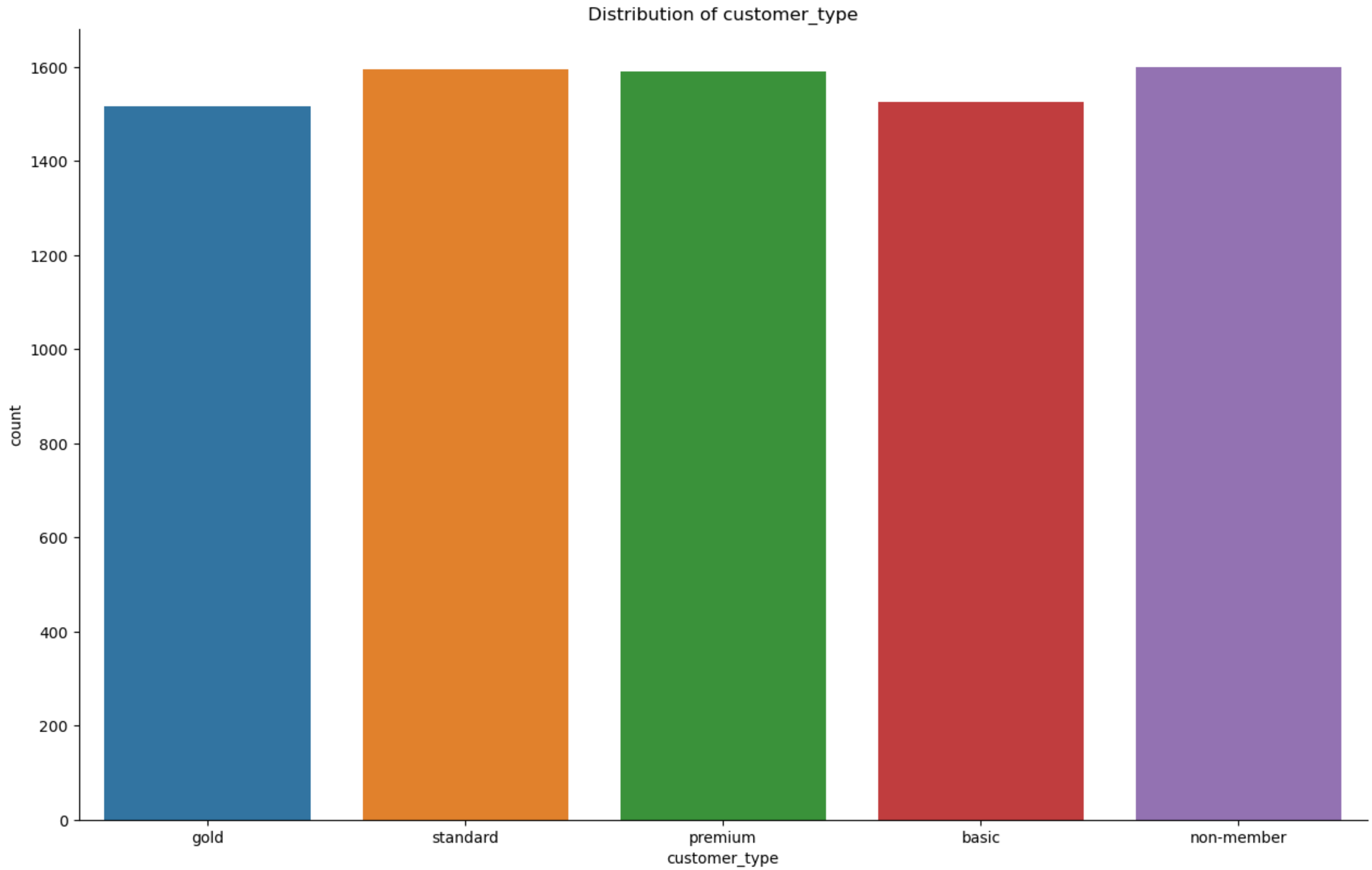
```
Requirement already satisfied: matplotlib in c:\users\asus\anaconda3\lib\site-packages (3.5.2)
Requirement already satisfied: pyparsing>=2.2.1 in c:\users\asus\anaconda3\lib\site-packages (from matplotlib) (3.0.9)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\asus\anaconda3\lib\site-packages (from matplotlib) (4.25.
0)
Requirement already satisfied: pillow>=6.2.0 in c:\users\asus\anaconda3\lib\site-packages (from matplotlib) (9.2.0)
Requirement already satisfied: cycler>=0.10 in c:\users\asus\anaconda3\lib\site-packages (from matplotlib) (0.11.0)
Requirement already satisfied: numpy>=1.17 in c:\users\asus\anaconda3\lib\site-packages (from matplotlib) (1.21.5)
Requirement already satisfied: packaging>=20.0 in c:\users\asus\anaconda3\lib\site-packages (from matplotlib) (21.3)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\asus\anaconda3\lib\site-packages (from matplotlib) (1.4.2)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\asus\anaconda3\lib\site-packages (from matplotlib) (2.
8.2)
Requirement already satisfied: six>=1.5 in c:\users\asus\anaconda3\lib\site-packages (from python-dateutil>=2.7->matplo
tlib) (1.16.0)
Requirement already satisfied: numpy in c:\users\asus\anaconda3\lib\site-packages (1.21.5)
```

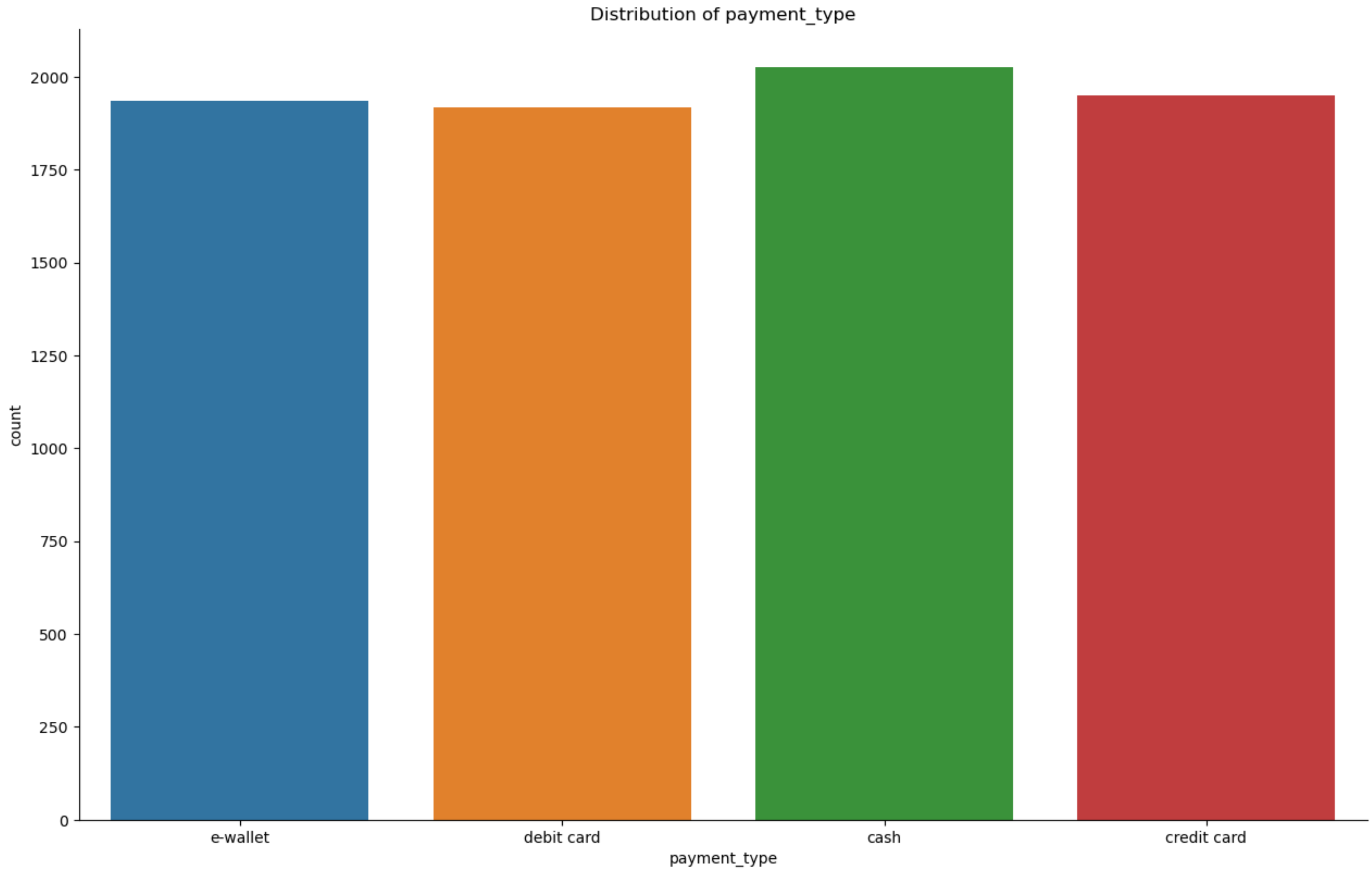
In [78]:
```
import matplotlib.pyplot as plt
import numpy as np
```

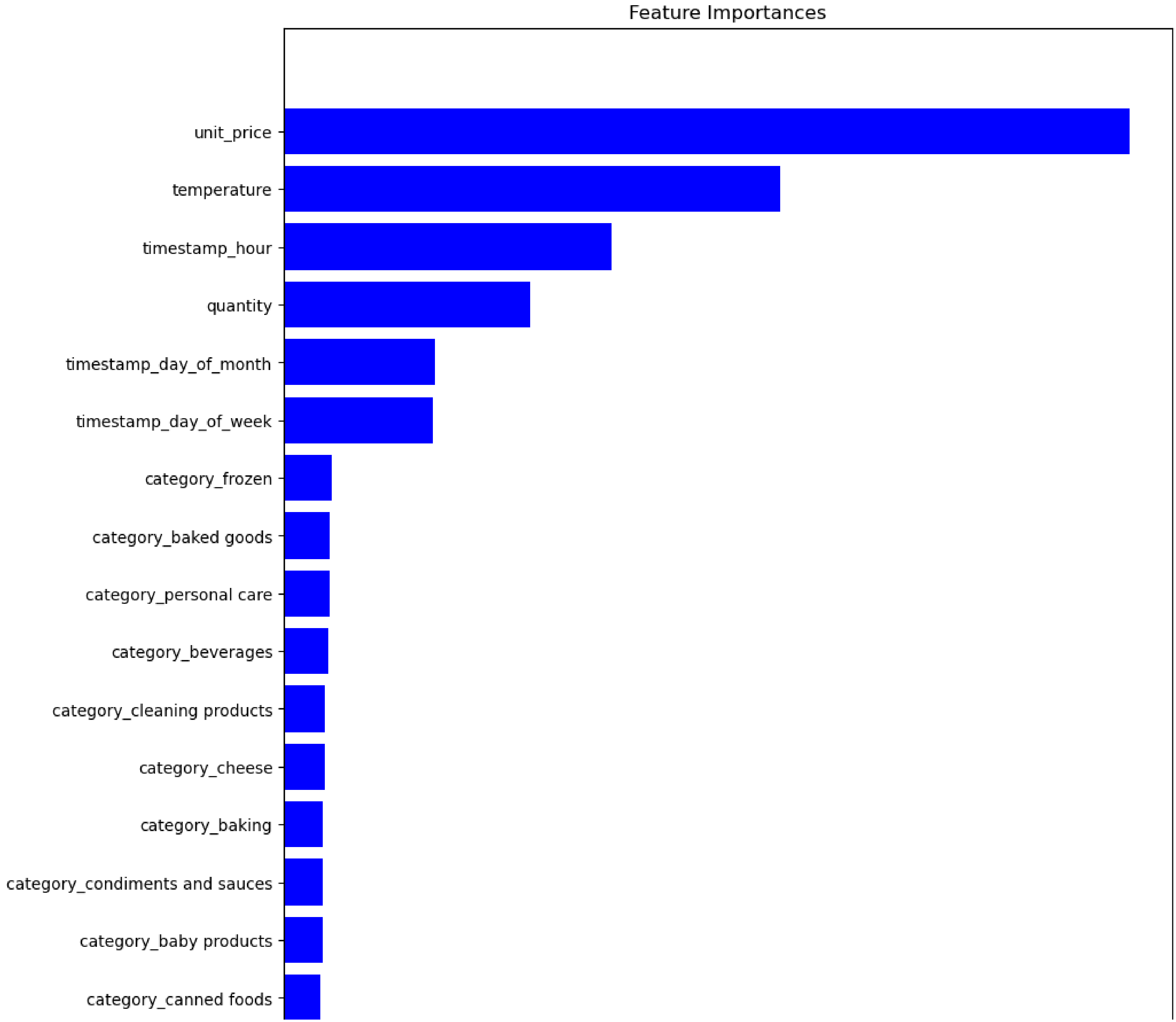
In [79]:
```
features = [i.split("__")[0] for i in X.columns]
importances = model.feature_importances_
indices = np.argsort(importances)
```

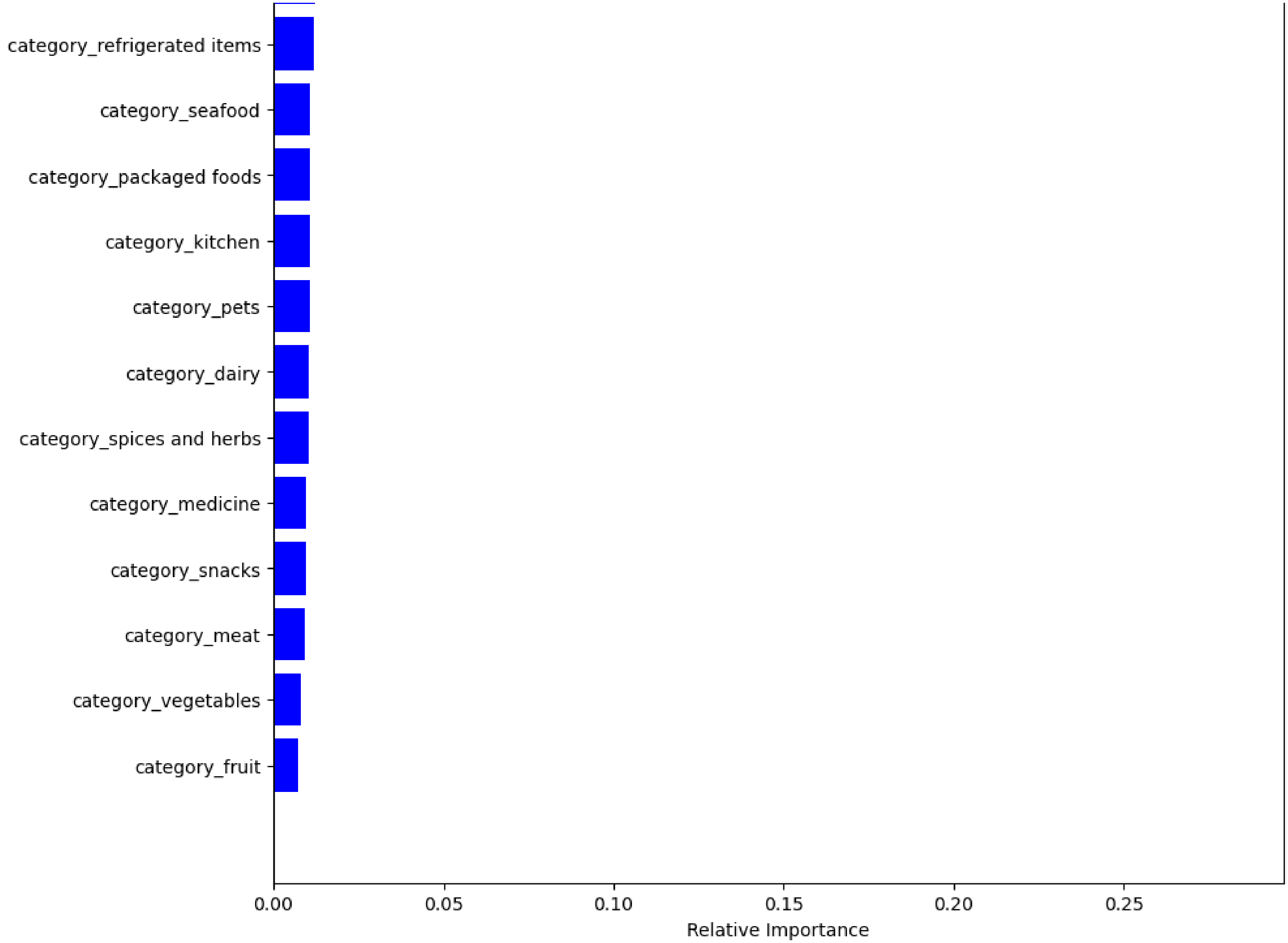
```
fig, ax = plt.subplots(figsize=(10, 20))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='b', align='center')
plt.yticks(range(len(indices)), [features[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```




Distribution of customer_type
count
1600
1400
1200
1000
800
600
400
200
0
gold
standard
premium
basic
non-member
customer_type


Distribution of payment_type
count
2000
1750
1500
1250
1000
750
500
250
0
e-wallet
debit card
cash
credit card
payment_type


Feature Importances
unit_price
temperature
timestamp_hour
quantity
timestamp_day_of_month
timestamp_day_of_week
category_frozen
category_baked goods
category_personal care
category_beverages
category_cleaning products
category_cheese
category_baking
category_condiments and sauces
category_baby products
category_canned foods



FINAL CONCLUSION

This feature importance visualisation tells us:

The product categories were not that important The unit price and temperature were important in predicting stock The hour of day was also important for predicting stock With these insights, we can now report this back to the business