Temasek POLYTECHNIC

SCHOOL OF INFORMATICS & IT

Machine Learning for Decision Making

Understanding what drives customers to purchase more products

Student Name (Matric Number):	Javen Lai Le Yu (2202934B)
Tutorial Group:	P03
Tutor:	Zaw Htet Wai
Submission Date:	12/11/2023

Declaration of Originality

I am the originator of this work and I have appropriately acknowledged all other original sources used as my references for this work.

I understand that Plagiarism is the act of taking and using the whole or any part of another person's work, including work generated by AI, and presenting it as my own.

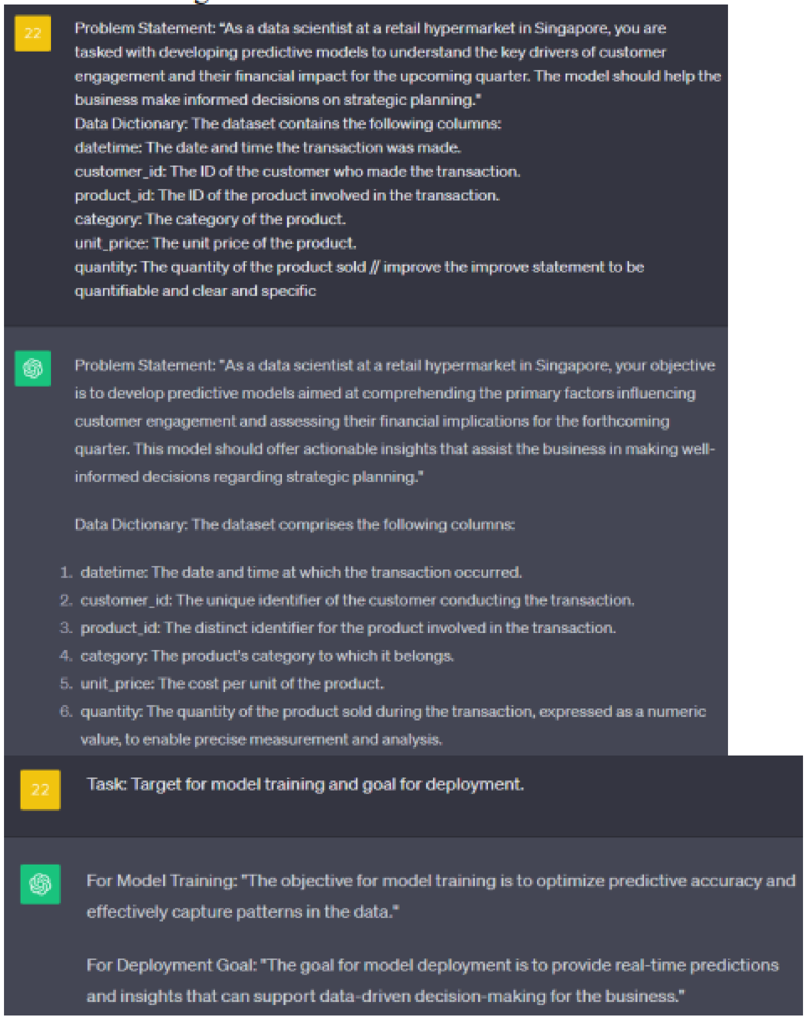
I understand that Plagiarism is an academic offence

and if I am found to have committed or abetted the offence of plagiarism in relation to this submitted work, disciplinary action will be enforced.

Describe how you have used Generative AI tools such as ChatGPT or Dall.E-2 in your assignment.

Show snapshots of the conversations with the AI tool (i.e., the prompts you used and the response you get from the AI tool).

1. Problem framing with ChatGPT:



22

Problem Statement: "As a data scientist at a retail hypermarket in Singapore, you are tasked with developing predictive models to understand the key drivers of customer engagement and their financial impact for the upcoming quarter. The model should help the business make informed decisions on strategic planning."
Data Dictionary: The dataset contains the following columns:
datetime: The date and time the transaction was made.
customer_id: The ID of the customer who made the transaction.
product_id: The ID of the product involved in the transaction.
category: The category of the product.
unit_price: The unit price of the product.
quantity: The quantity of the product sold // improve the improve statement to be quantifiable and clear and specific

Problem Statement: "As a data scientist at a retail hypermarket in Singapore, your objective is to develop predictive models aimed at comprehending the primary factors influencing customer engagement and assessing their financial implications for the forthcoming quarter. This model should offer actionable insights that assist the business in making well-informed decisions regarding strategic planning."

Data Dictionary: The dataset comprises the following columns:

1. datetime: The date and time at which the transaction occurred.
2. customer_id: The unique identifier of the customer conducting the transaction.
3. product_id: The distinct identifier for the product involved in the transaction.
4. category: The product's category to which it belongs.
5. unit_price: The cost per unit of the product.
6. quantity: The quantity of the product sold during the transaction, expressed as a numeric value, to enable precise measurement and analysis.

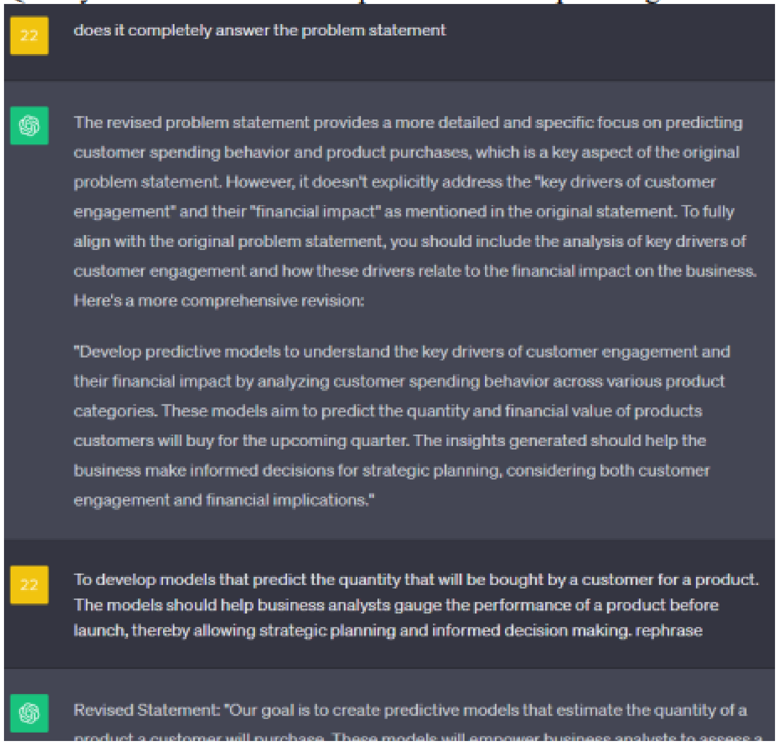
22

Task: Target for model training and goal for deployment.

For Model Training: "The objective for model training is to optimize predictive accuracy and effectively capture patterns in the data."

For Deployment Goal: "The goal for model deployment is to provide real-time predictions and insights that can support data-driven decision-making for the business."

2. Quality check + Grammar improvements and phrasing to make explanations fluent:



22 does it completely answer the problem statement

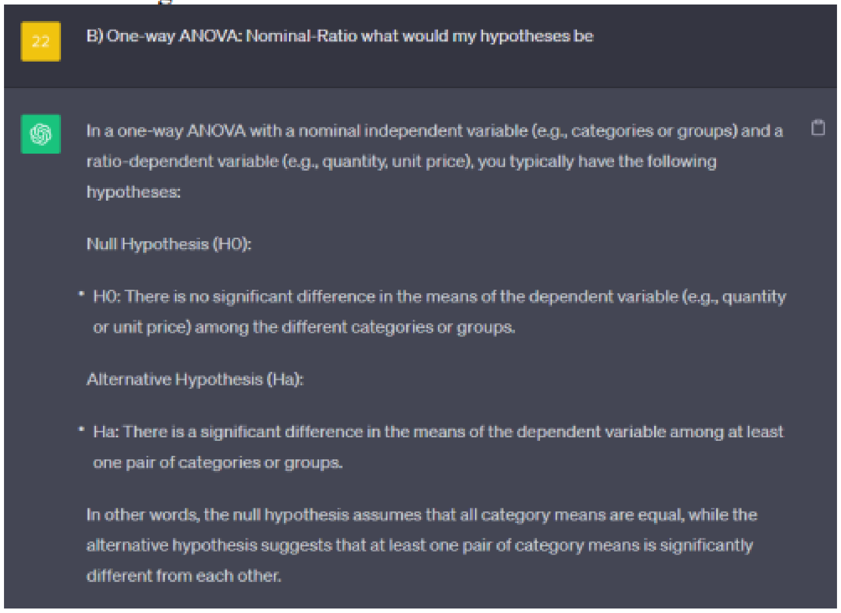
The revised problem statement provides a more detailed and specific focus on predicting customer spending behavior and product purchases, which is a key aspect of the original problem statement. However, it doesn't explicitly address the "key drivers of customer engagement" and their "financial impact" as mentioned in the original statement. To fully align with the original problem statement, you should include the analysis of key drivers of customer engagement and how these drivers relate to the financial impact on the business. Here's a more comprehensive revision:

"Develop predictive models to understand the key drivers of customer engagement and their financial impact by analyzing customer spending behavior across various product categories. These models aim to predict the quantity and financial value of products customers will buy for the upcoming quarter. The insights generated should help the business make informed decisions for strategic planning, considering both customer engagement and financial implications."

22 To develop models that predict the quantity that will be bought by a customer for a product. The models should help business analysts gauge the performance of a product before launch, thereby allowing strategic planning and informed decision making. rephrase

Revised Statement: "Our goal is to create predictive models that estimate the quantity of a

3. Fact checking:



22 B) One-way ANOVA: Nominal-Ratio what would my hypotheses be

In a one-way ANOVA with a nominal independent variable (e.g., categories or groups) and a ratio-dependent variable (e.g., quantity, unit price), you typically have the following hypotheses:

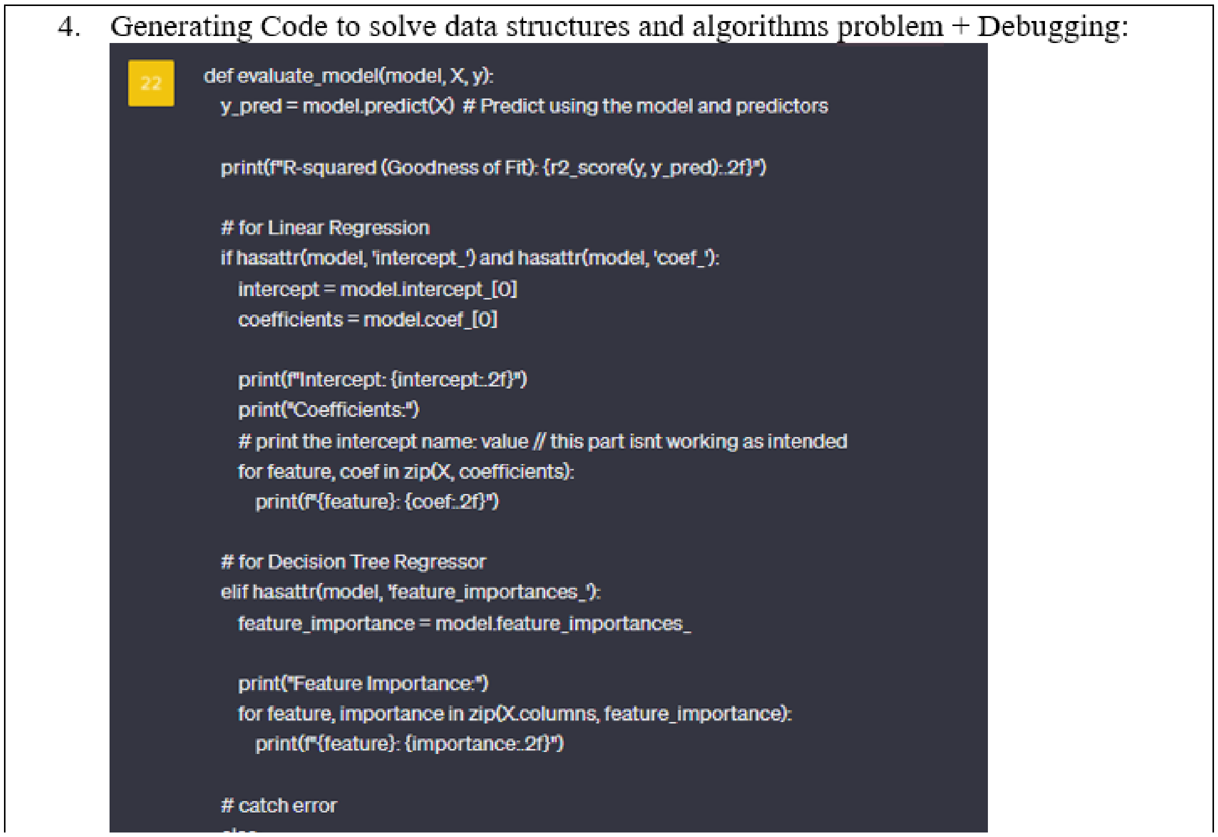
Null Hypothesis (H0):

- H0: There is no significant difference in the means of the dependent variable (e.g., quantity or unit price) among the different categories or groups.

Alternative Hypothesis (Ha):

- Ha: There is a significant difference in the means of the dependent variable among at least one pair of categories or groups.

In other words, the null hypothesis assumes that all category means are equal, while the alternative hypothesis suggests that at least one pair of category means is significantly different from each other.



4. Generating Code to solve data structures and algorithms problem + Debugging:

```
def evaluate_model(model, X, y):
  y_pred = model.predict(X)  # Predict using the model and predictors

  print(f"R-squared (Goodness of Fit): {r2_score(y, y_pred):.2f}")

  # for Linear Regression
  if hasattr(model, 'intercept_') and hasattr(model, 'coef_'):
    intercept = model.intercept_[0]
    coefficients = model.coef_[0]

    print(f"Intercept: {intercept:.2f}")
    print("Coefficients:")
    # print the intercept name: value // this part isnt working as intended
    for feature, coef in zip(X, coefficients):
      print(f"{feature}: {coef:.2f}")

  # for Decision Tree Regressor
  elif hasattr(model, 'feature_importances_'):
    feature_importance = model.feature_importances_

    print("Feature Importance:")
    for feature, importance in zip(X.columns, feature_importance):
      print(f"{feature}: {importance:.2f}")

  # catch error
```

1. Introduction

Business Context and Requirement

Retail Hypermarket is a Singapore-based store aiming to build strong customer relationships to secure long-term success. They want to enhance customer engagement and plan to launch a sales campaign in the upcoming quarter. However, they are uncertain about the factors that drive customers to make more purchases. As a data scientist, my role is to uncover the key drivers of customer engagement and their financial impact, so that I can provide data-driven sales strategies that can be backed by historical customer transactions to be highly successful if implemented.

1.1 Problem Statement

"Develop predictive models to understand the key drivers of customer engagement and their financial impact for the upcoming quarter. The model should assist the business to make informed decisions on strategic planning."

1.2 Objective

Develop machine learning models to help my client understand how various factors, such as product type, pricing, and market trends for different quarters, influence the quantity of products purchased by different customers. These insights will enable the client to create data-driven sales strategies that make customers buy more products, thereby increasing customer spending and generating more revenue for the company. These strategies will be implemented in the next quarter as part of a sales campaign to improve sales revenue by increasing the average customer purchase volume.

The models will be trained to **predict how many products a customer bought (Quantity) for a transaction** based on customer type and features related to the product, market trends, and quarter. Quantity of a customer's purchase is the metric to quantify customer engagement. Increasing customer engagement = increasing customer purchase volume, which leads to more products being sold hence more revenue for the store.

1.3 Goal for deployment

Understand which market trends and pricing for specific product types lead to the highest customer engagement among various customer types. This will enable the formulation of data-driven strategies to encourage customers to buy more products in the next quarter.

1.3.1 Whitebox:

- Highly interpretable
- Rationale behind how the model derives a prediction can be clearly understood.

1.3.2 Blackbox:

- Predictions are highly accurate.
- Rationale behind how the model derives its prediction can be understood.

1.4 Inputs for Models

Target: Quantity

Potential predictors:

- Customer type: To identify the different types of customers. Since different customer types could have different behavioral patterns, this could be a relevant feature for the machine to learn.
- Product type: Certain products are meant for bulk purchases by nature. Example: Stationeries like pen, pencil, and eraser. Hence the 'ideal sales' for different products vary and products need to be compared separately.
- Product pricing: Customers may get tempted to buy more when there is a discount, hence the pricing is a potential feature for the machine to learn the pricing patterns for different products, so that a pricing strategy to boost customer engagement could be engineered by using the model.
- Quarter: To capture market trends for different quarters of the year.

1.5 Considerations

- Predictors should be readily available before the occurrence of prediction to ensure the model is usable for predicting.
- Data used to train models should be recent, to ensure the patterns and intricacies the model works on match the current trend. This makes the model usable as the predictions are reliable.
- Models should be well-generalized to predict unseen future occurrences with similar accuracy as the training dataset; no overfitting on training data.
- Models should adhere to AI ethics and regulations, ensuring transparency, fairness, absence of bias, and non-discrimination.

1.6 Metrics for Model Evaluation

1.6.1 **Goodness of Fit: R^2** to analyse how well the predictions can be explained by the predictor values, to assess if the patterns and intricacies of how customers engage in purchase of various products have been captured by the model.

1.6.2 **Accuracy: Mean Absolute Error (MAE)** to assess the error of an average prediction; how far an average prediction deviates from the actual value. This metric is chosen for its simplistic interpretation and appropriateness for the Target's nature.

> Root Mean Squared Error (RMSE) is not chosen due to its slightly complex interpretation and nature of amplifying small errors due to the squared term. As the quantity range is quite small ranging from 1 to 20, there is no need to penalize larger errors as outliers are an impossibility.

1.7 Success Criteria for Project

- Deploy **1** predictive model to be presented to Retail Hypermarket for usage.

- Before deployment, a model's MAE should be 2 or below to ensure highly accurate predictions with an average deviation from the truth of no more than 2 units. This is because the Quantity/engagement of a transaction only has a range of 1 to 20, which is very small. Hence, the margin for error should be relative to this range.

> MAE of 0 is the best-case scenario. However, this may not be realistically achievable. MAE of 0 may also indicate overfitting and should checked and confirmed that the model is not overfitted before deployment.
>
> A large MAE makes the model unreliable for predicting customer product purchases, casting doubt on the accuracy of the insights derived from the model.

- Model fit (R-Squared value) should be above 80%, to affirm model is explainable and its decisions are closely based on the truth of historical data; no underfitting, hence reliable.
- Goal of deployment (1.3) should be fulfilled.
- The black-box model should outperform the white-box model by a noticeable margin of minimally 5% if chosen for deployment, to ensure the choice of the more complex model is justified by significantly improved accuracy.

1.8 Environment

- New Products: I should refrain from adding precise details about a product, such a Product ID, to allow the models to forecast engagement on new products that exhibit similar qualities with existing products within the training dataset. For example, the new product is in an existing category. If the new product is in a newly created category not in training dataset, the model cannot be used for predicting customer engagement on that product. This ensures the usability of a model is sustainable for future usage and does not require frequent updates.
- New Customers: The models should able to predict the customer engagement for customer not within training dataset. This ensures the models are versatile and usable in real world scenario.

> This means I cannot use features that specify the exact customer. For example, name or customer ID. This is also to avoid overfitting.

- Economic Status: Customer interests and spending behaviour may change during recessions or economic boom. These trends may not be captured and identified if they are not within the training dataset.
- Market Trends: The market trend and customers' spending behavior can change rapidly in this fast-paced society, and may differ from the training data.
- Seasonal changes: Unprecedented events like a virus outbreak, unexpected intense rainfall or scorching weather period, may deter customers from visiting Retail Hypermarket, resulting financial performance that cannot be explained by the models.
- Competition: New retail companies may emerge in the future and influence customers' decision on whether to patronize Retail Hypermarket, and the models are unable to account for such external factors.

In summary:

1. The training dataset must be recent and reflective of current trends and customer behaviour.
2. The Model's code and documentations should be clear, easy to understand, and adaptable for future modifications (e.g. changing dataset, random_state).

1.9 Target leakage

Do not include:

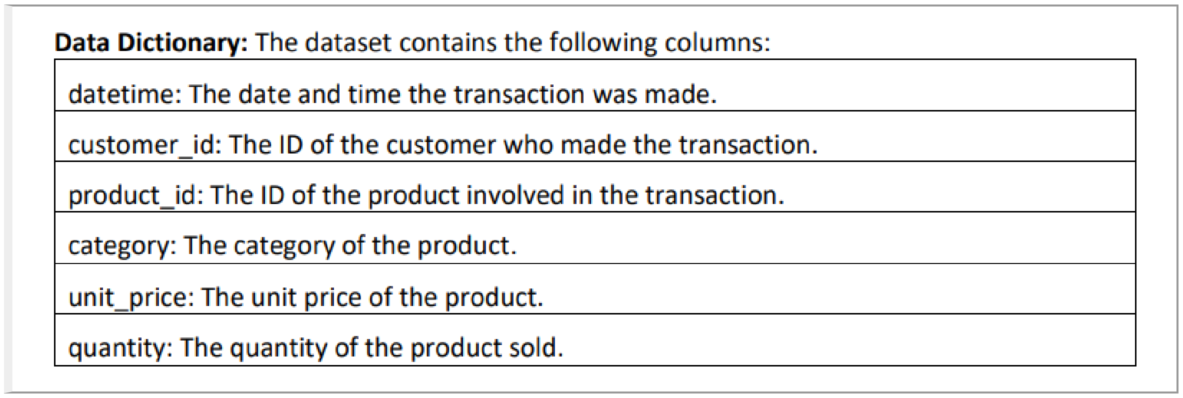
- **Features not available at the time of prediction** as this makes the model unusable as the business would not have access to such information when making pricing decisions. E.g. Year.
- **Features directly related unit price** to prevent multicollinearity issues.
- **Features with unrealistically high collinearity with unit price**, which could be derived from the target and bias the model.
- **High cardinality columns** like ID to prevent overfitting and ensure model generalization with new, unseen data.
- **Multicollinearity should be avoided** as it leads to reduced model interpretability, unstable coefficient estimates,

2. Data Attributes

2.1 Data Understanding

The dataset 'synthetic_data.csv' was provided by Retail Hypermarket, which contains the company's transaction records from 2022 onwards. A data dictionary, attached below, has also been provided for understanding what each column represents.

The dataset should be trusted as it was received directly from the client.



Data Dictionary: The dataset contains the following columns:

datetime: The date and time the transaction was made.
customer_id: The ID of the customer who made the transaction.
product_id: The ID of the product involved in the transaction.
category: The category of the product.
unit_price: The unit price of the product.
quantity: The quantity of the product sold.

Evaluation:

1. Customer Type is not given, but Customer ID is provided. Perhaps the Customer Type could be in the ID?
2. Category has similar meaning to Product Type where it provides detail about what the product is. Hence, category could represent the product type.
3. Product price is provided as unit_price.
4. Features related to market trend are absent from dataset. However, potential features like Day, Month, and Quarter could be derived from datetime to allow the model to capture market trends for different periods of the year.

2.2 Data Inspection

2.2.1 Importing libraries

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

SEED = 2202934 # admin number for random_state
```

2.2.2 Loading csv into DataFrame

In [2]:
```
df = pd.read_csv('synthetic_data.csv')
print("Number of observations: ", len(df)) # count rows in df
```

```
Number of observations:  331664
```

There is an abundance of data. The 300K+ rows of data is enough for training and testing the models.

2.2.3 Exploring Dataset

In [3]:
```
pd.set_option('display.float_format', '{:.10f}'.format) # show full unit_price without truncation
df.head()
```

Out[3]:

	Datetime	Product_ID	Category	Quantity	Unit_Price	Customer_ID
0	2022-01-07	10106959	Stationery	9	1.5798753892	a225207859
1	2022-01-09	90097406	Sports	4	196.2533774599	a225207859
2	2022-01-10	10010465	Electronics	1	825.3742907058	a225207859
3	2022-01-14	10010510	Electronics	2	325.9650346646	a225207859
4	2022-01-16	40049430	Books	1	22.6019194627	a225207859

In [4]:
```
df.tail()
```

Out[4]:

	Datetime	Product_ID	Category	Quantity	Unit_Price	Customer_ID
331659	2022-12-18	20026116	Groceries	5	5.3781616834	c891387366
331660	2022-12-23	10018925	Electronics	1	1193.8285186999	c891387366
331661	2022-12-27	20022820	Groceries	3	5.1754917125	c891387366
331662	2022-12-27	40042985	Books	2	38.5985362367	c891387366
331663	2022-12-31	20025163	Groceries	3	5.3977440791	c891387366

Evaluation:

1. **Datetime is missing time**. There is nothing that can be done about missing time as I am only provided with this dataset.
2. There is **no columns like Quarter or Month**. However, these seasonality features can be derived from the date in Datetime.
3. Unit_Price is a non-terminating number, which is strange as its unconventional for prices at supermarkets to go beyond cents. Hence, **Unit_Price should be rounded off to 2 d.p.**
4. The records seem to be in **time-series**, and ends at 31 December 2022? (Continued at 2.3.2.b)

2.2.4 Are there duplicated records?

In [5]:
```
df.duplicated(subset=['Datetime', 'Customer_ID']).sum() # sum up number of duplicated rows
```

Out[5]:
```
29362
```

This indicates that **there are customers who make multiple transactions a day**, indicating that they bought different products at the same time.

To check if there are duplicated records, I should factor in Product_ID as Retail Hypermarket wouldn't seperate the same product into different transaction.

In [6]:
```
df.duplicated(subset=['Datetime', 'Customer_ID', 'Product_ID']).sum() # sum up duplicated transacti
```

Out[6]:
```
0
```

There are **no duplicated records in dataset**.

2.2.5 Check for data types and missing values

In [7]:
```
pd.set_option('display.float_format', '{:.3f}'.format) # revert df to round of to 3 d.p.
```

In [8]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 331664 entries, 0 to 331663
Data columns (total 6 columns):
 #   Column       Non-Null Count   Dtype  
---  ------       --------------   -----  
 0   Datetime     331664 non-null  object 
 1   Product_ID   331664 non-null  int64  
 2   Category     331664 non-null  object 
 3   Quantity     331664 non-null  int64  
 4   Unit_Price   331664 non-null  float64
 5   Customer_ID  331664 non-null  object 
dtypes: float64(1), int64(2), object(3)
memory usage: 15.2+ MB
```

Takeaways:

1. No missing values for any columns because all columns have 331, 664 non-null values.
2. Datetime column is not in its proper format of a date datatype.

2.2.6 Summary statistics

In [9]:
```
df.describe().loc[['min', 'mean', 'max']]
```

Out[9]:

	Product_ID	Quantity	Unit_Price
min	10010000.000	1.000	1.001
mean	30263284.237	9.125	124.090
max	90099999.000	20.000	1199.947

In [10]:
```
df.nunique() # count of unique values in a column
```

Out[10]:
```
Datetime          368
Product_ID      75781
Category           10
Quantity           20
Unit_Price     331664
Customer_ID      2648
dtype: int64
```

In [11]:
```
df.Category.unique() # identify all unique values in category col
```

Out[11]:
```
array(['Stationery', 'Sports', 'Electronics', 'Books', 'Groceries',
       'Health & Beauty', 'Furniture', 'Automotive', 'Clothing', 'Toys'],
      dtype=object)
```

Takeaways:

1. All Product_IDs are 8 char.
2. Every transaction can only involve 1 to 20 of the same product.
3. Cheapest product is 1 dollar and most expensive is 1200 dollars. An average product costs 124 dollars per unit.
4. There are 10 types of product Categories.

Evaluation:

1. Datetime should not contain any anomalies if it can be successfully parsed into Date datatype.
2. Identifying anomaly values for Product_ID and Customer_ID can be tough as there are thousands of unique values; too many to manually inspect. Hence, I shall perform Exploratory Data Analysis to investigate for irregularities in these columns.
3. Category does not have any anomalies can all unique values are logical for a product category.
4. Quantity has no anomalies as all unique values are integer and within range of 1 to 20; no irregular values like 0.2 quantity or -1 quantity. This also suggests that there can be no partial purchases or refunds.

2.2.7 Do Products have a fixed unit price

In [12]:
```
Product = df.sort_values(by=['Product_ID'])
Product.head(4)
```

Out[12]:

	Datetime	Product_ID	Category	Quantity	Unit_Price	Customer_ID
126708	2022-05-22	10010000	Electronics	2	850.915	a704811449
326206	2022-05-26	10010000	Electronics	2	357.381	c567078712
173955	2022-09-06	10010000	Electronics	1	495.939	c849466466
105432	2022-11-07	10010000	Electronics	1	791.225	d703549797

Analysis:

Price of product changes over time. This indicates that the store practices competitive pricing by changing prices to offer attractive deals that encourage customers to engage in purchases.

2.3 Data Cleaning

2.3.0 Rectifications based on Data Inspection:

1. Round off Unit_Price to 2 d.p.
2. Parse Datetime into date datatype.
3. Features extraction on Date to create Quarters.

2.3.1 Unit_Price:

In [13]:
```
df['Unit_Price'] = round(df['Unit_Price'], 2)
```

2.3.2.a Parsing Datetime:

In [14]:
```
df['Datetime'] = pd.to_datetime(df['Datetime'])
```

There are no anomalies in Datetime as all values conform to Date datatype format.

2.3.2.b Confirmation in data is in time-series

In [15]:
```
df.sort_values(by='Datetime', inplace=True)
df.tail(8)
```

Out[15]:

	Datetime	Product_ID	Category	Quantity	Unit_Price	Customer_ID
48824	2023-01-03	50056627	Furniture	1	927.850	a793237418
150571	2023-01-03	70070603	Toys	9	24.140	d407005726
151609	2023-01-03	40041605	Books	3	5.020	a166502882
77889	2023-01-03	90093200	Sports	2	33.040	a253267567
200653	2023-01-03	70077230	Toys	3	8.880	a731741739
115899	2023-01-03	50050749	Furniture	2	985.720	a555546365
229742	2023-01-03	10012069	Electronics	1	1112.050	d112052977
95489	2023-01-03	30030537	Clothing	5	24.710	b004263214

Analysis:

- It's strange that these transactions were placed randomly within the 2022 dataset rather than at the end of the dataset as expected.
- This raises concerns about the reliable of the Datetime column, especially when the time values are missing from this column.

Evaluation:

- The Datetime for 2023 records may be incorrect as these records do not follow the typical pattern of transactional record systems where new transactions are found at the end of the dataset.
- Since there are 300K rows of data, sufficient for training and validating the model, I shall exclude the 2023 records and train my model solely on 2022 data due to suspicions in reliability of 2023 records: because time is missing from Datetime, its possible that this column has problems.
- However, 2023 records could be used for testing of models.

In [16]:
```
df_test = df[df['Datetime'].dt.year == 2023]
df = df[df['Datetime'].dt.year != 2023]
```

In [17]:
```
len(df_test)
```

Out[17]:
```
386
```

2.3.3 Quarter:

Customer spending behavior can varies across seasons, hence I will create this column to allow user to predict prices for different quarters of the year.

In [18]:
```
df['Quarter'] = df['Datetime'].dt.to_period('Q').astype(str).str[-1]
# Only need the Quarter number, hence index last number in Quarter
```

2.4 Exploratory Data Analysis (EDA)

To delve into relationships between features and the Target, to identify potential data issues and whether the feature is relevant for predicting quantity.

2.4.1 Examine df:

```
In [19]: df.sample(12) # randomly sample df to analyse data
```

Out[19]:

	Datetime	Product_ID	Category	Quantity	Unit_Price	Customer_ID	Quarter
207951	2022-11-11	20029539	Groceries	14	9.190	a906344165	4
330477	2022-04-21	20029411	Groceries	15	9.810	c304909862	2
38818	2022-07-05	70074097	Toys	6	39.810	b061313184	3
143786	2022-05-13	60060465	Automotive	4	78.120	c771987774	2
109733	2022-10-31	60067062	Automotive	2	67.140	c711463336	4
123346	2022-09-08	20027464	Groceries	18	15.360	a249419459	3
168299	2022-03-12	20021655	Groceries	18	10.220	c711191185	1
68837	2022-01-06	10105060	Stationery	1	7.850	d844462141	1
24895	2022-10-13	20024251	Groceries	16	15.320	a161149597	4
243090	2022-09-28	20026359	Groceries	14	18.260	b077951315	3
57074	2022-02-22	20024083	Groceries	10	11.010	c392301422	1
316594	2022-10-30	20028476	Groceries	15	18.440	d550552576	4

2.4.2 EDA Graph Plotter:

```
In [20]: def eda_plot(df, category, measure, plot_type='line', measurement='sum'):
             category_total = df.groupby(category)[measure].agg(measurement)      # Aggr the measure per cate
             plt.figure(figsize=(16, 6))                                          # Set the size of the grapl

             # Toggle to the selected chart
             if plot_type == 'line':
                 plt.plot(category_total.index, category_total.values, marker='o', linestyle='-')
                 plt.grid(True)                                                   # turn on grid for easy rea

             elif plot_type == 'bar':
                 category_total = category_total.sort_values(ascending=False)     # Sort the data in descend
                 plt.bar(category_total.index, category_total.values)

             else:
                 print("Invalid plot_type. Please use 'line' or 'bar'.")          # error tolerance
                 return

             # Remove '_' for easier readibility of Legends
             measure = measure.replace('_', ' ')
             category = category.replace('_', ' ')

             # Add labels above the data points
             for x, y in zip(category_total.index, category_total.values):
                 plt.text(x, y, f'{y:.2f}', ha='center', va='bottom')

             plt.title(f'{measurement.capitalize()} {measure} per {category}')
             plt.xlabel(category)
             plt.ylabel(measure)
             plt.xticks(category_total.index)
             plt.show()
```

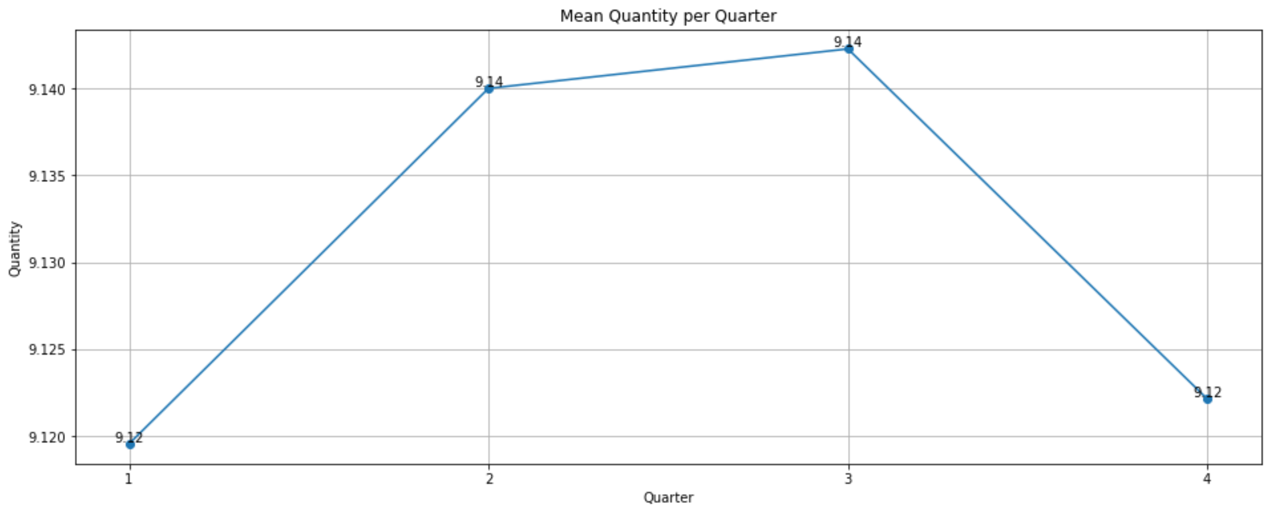
Usage:

> eda_plot(DataFrame, [Column in df], [measure, must be numerical], plot_type=[Type of Plot, line or bar, default=line], measurement=[measurement, sum/mean/median. default=sum])

2.4.3.a Engagement by Quarter

- Is there a discernable customer engagement pattern for each quarter?

```
In [21]: eda_plot(df, 'Quarter', 'Quantity', measurement='mean') # how much products an average customer buys
```



A) Analysis:

- On average, a customer purchases 9 products per transaction.
- Average engagement of a customer is indifferent for all quarters; only small and insignificant different of 0.02 units.

B) Conclusion:

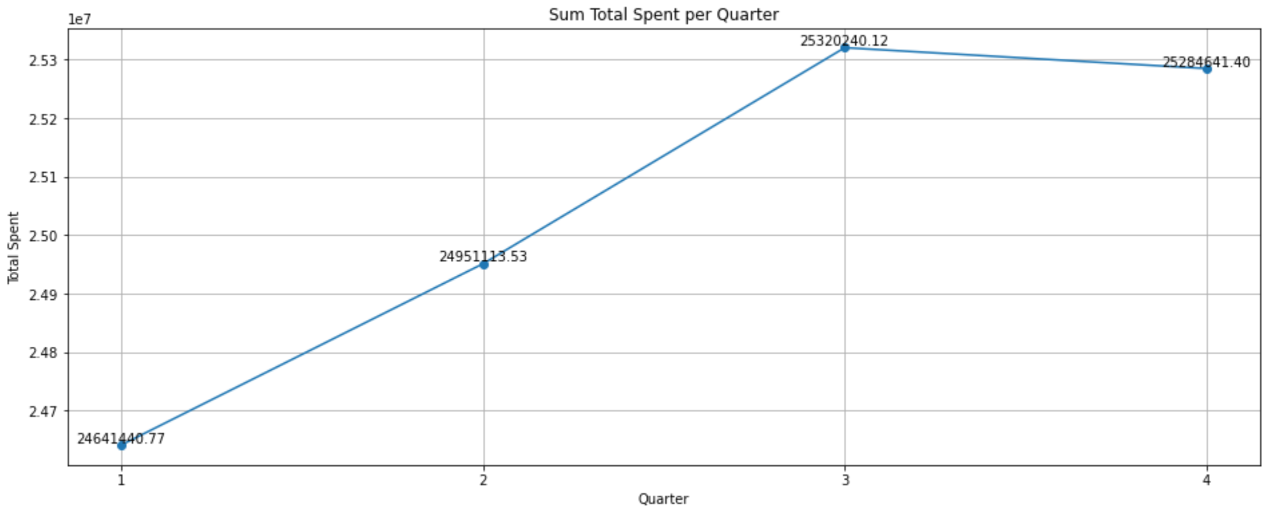
An average customer engages equally in a purchase throughout the year;

The average customer engagement for all quarters are the same with no significant variation.

2.4.3.b Market Trend by Quarter

- Is there an underlying market trend for each quarter?

```
In [22]: df['Total_Spent'] = df['Unit_Price'] * df['Quantity']      # calculate to total spending per transac
         eda_plot(df, 'Quarter', 'Total_Spent', measurement='sum')  # total spending per Quarter
```



A) Analysis:

The customer engagement for Retail Hypermarket during Q1 and Q2 is lower than the engagement during Q3 and Q4, as the total spendings of customers during Q1 and Q2 are below 25,000,000 dollars.

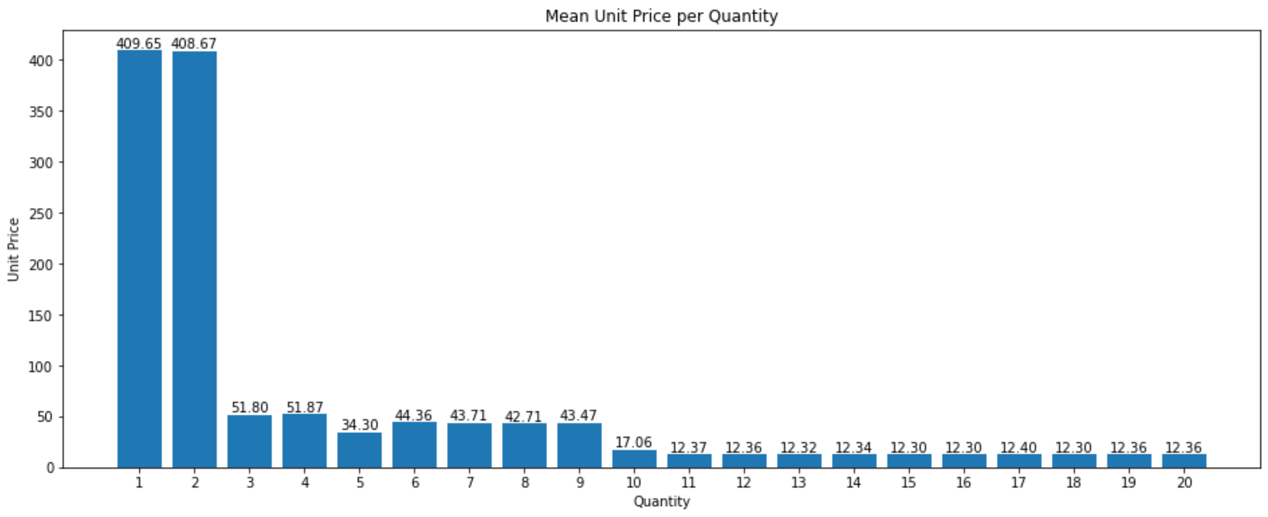
B) Conclusion:

This suggests that there could be an underlying market trend there are **more customers** who buy more and **spend more during the later half of the year**.

2.4.4 Relationship between quantity and quality

- Do customers purchase more when products are cheaper or buy fewer when prices are higher?

```
In [23]: eda_plot(df, 'Quantity', 'Unit_Price', 'bar', measurement='mean') # average customer engage for dif
```



A) Analysis:

1. Each transaction can have a quantity of 1 to 20 of a product.
2. 3 Classes observed:
    - Customers usually only buy 1-2 of an expensive product.
    - Customers buy 3-9 of a mid-range priced product.
    - Customers buy cheap products below 20 dollars in bulks of 10-20.

B) Evaluation:

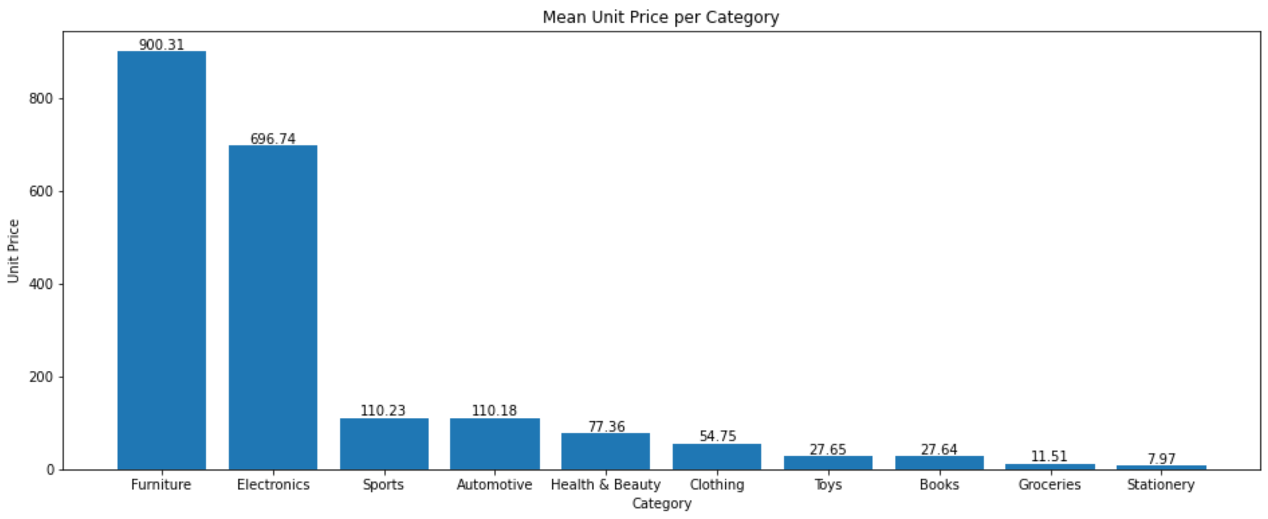
- Discounts isn't the explanation for customers buying more of a product as a discount of 88% (350/400) is illogical.
- Expensive products have poor customer engagement as customers only buy 1-2 of such products, while cheaper products below 50 dollars have higher engagement as customers buy such products in bulk purchases.

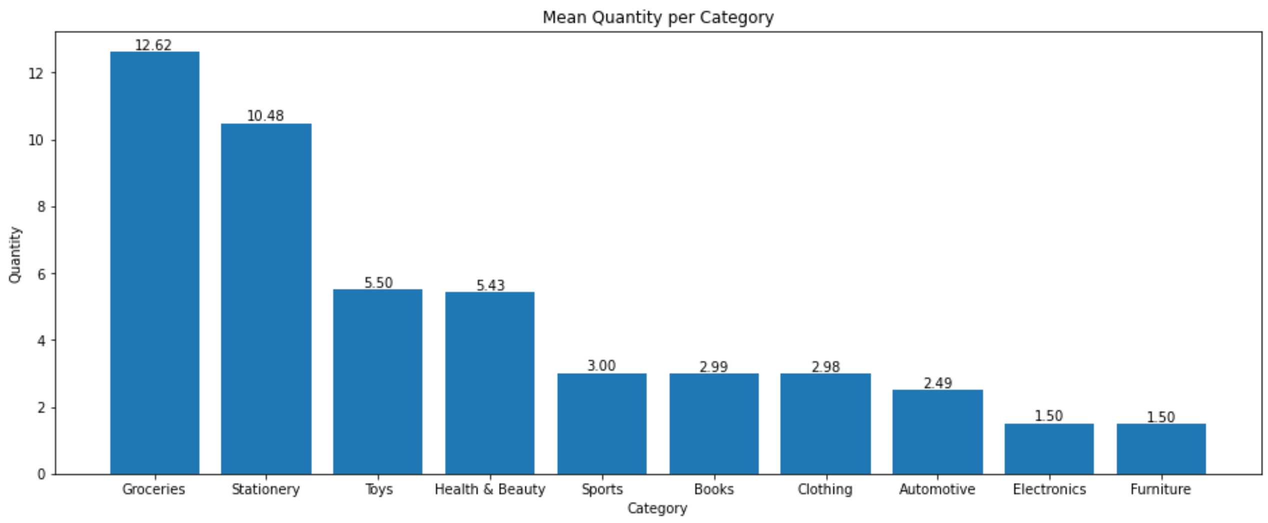
C) Conclusion:

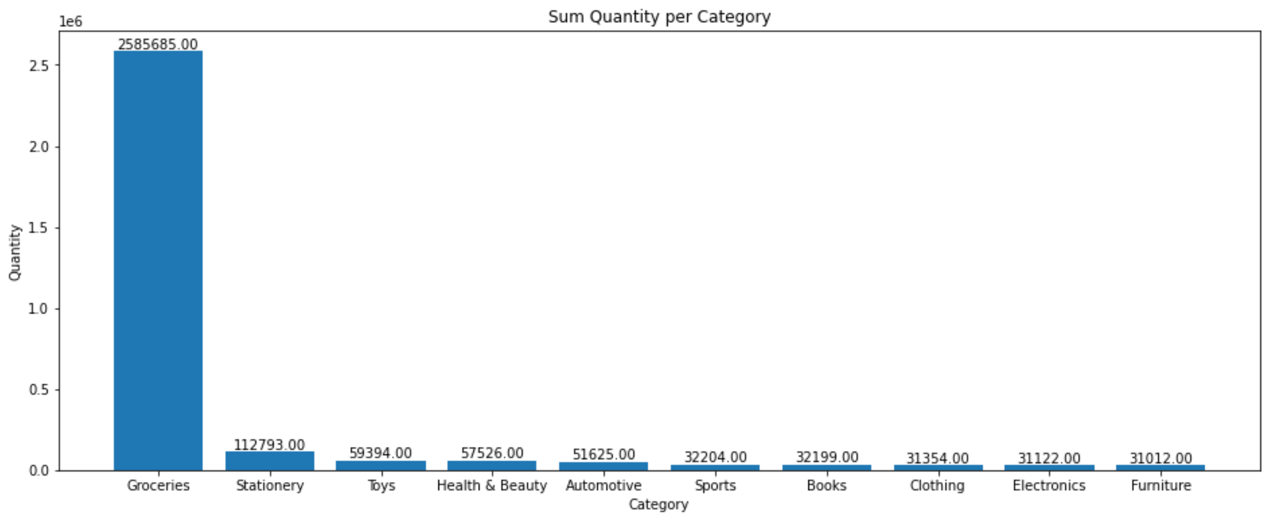
- Unit Price is a potential predictor because there is a distinguishable engagement pattern between cheap, mid-range, and expensive products.

2.4.5 Engagement across different product categories

```
In [24]: eda_plot(df, 'Category', 'Unit_Price', 'bar', 'mean') # average cost of a product
         eda_plot(df, 'Category', 'Quantity', 'bar', 'mean')   # how much an average customer purchases a pro
         eda_plot(df, 'Category', 'Quantity', 'bar', 'sum')    # total engagement per category
```







Calculating percentage of grocery purchases:

In [25]:
```
# Sum quantity for all rows where category is groceries
total_groceries = sum(df[df['Category'] == 'Groceries']['Quantity'])

# calculate percentage of purchases that groceries
total_groceries / sum(df['Quantity'])
```

Out[25]:
```
0.8547962024705495
```

A) Analysis

- 85% of purchases are products under Groceries category.
- Products from expensive categories like Furniture and Stationery have the poorest customer engagement as customers only purchase 1-2 of such products per transaction.
- Products from cheap categories like Groceries and Stationery have high customer engagement as customers purchase 10-13 of these products at a time.

B) Evaluation

- Customers treat Retail Hypermarket as a grocery store as most products bought are groceries.
- Pricier products tend to get lesser customer engagement. This could be due to:
  1. Nature of products: customers **do not need multiple furnitures**.
  2. Unaffordability: an **average customer cannot afford to spend 7000 dollars** to buy 10 electronic devices at a time.
- It's worth noting that there may be multicollinearity between the quantity of items purchased and the product category since the engagement patterns complement each other. This will be **investigated in 2.5**.

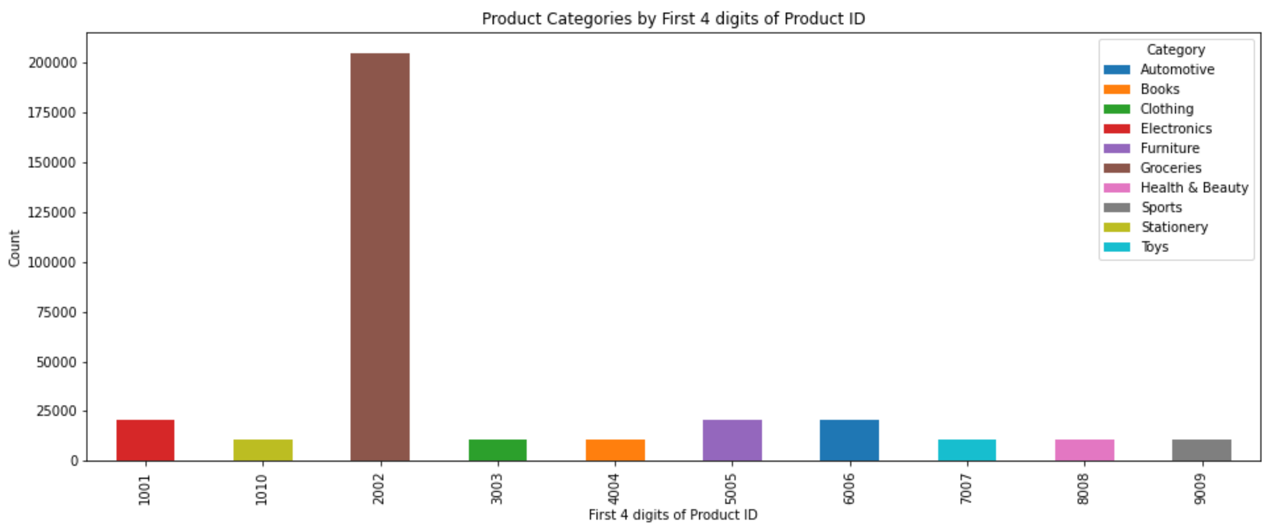
C) Conclusion

Category is a useful predictor because there is an observable pattern for customer engagement across different categories.

2.4.6.a Exploring relationship between Category and Product_ID

From 2.4.1, I discovered that the first 4 digits could represent something. To clarify my assumption, I shall investigate through visualisation.

In [26]:
```
# Extract the first digit from 'Product_ID' and create a new column 'First_Digit'
df['4_digits'] = df['Product_ID'].astype(str).str[0:4]

# Group data by 'First_Digit' and 'Category' and count the occurrences
grouped = df.groupby(['4_digits', 'Category']).size().unstack(fill_value=0)

# Create a bar plot to visualize the relationship
grouped.plot(kind='bar', stacked=True, figsize=(16, 6))
plt.title("Product Categories by First 4 digits of Product ID")
plt.xlabel("First 4 digits of Product ID")
plt.ylabel("Count")
plt.show()
```



In [27]:
```
df.Product_ID.nunique() # count of distinct values
```

Out[27]:
```
75725
```

A) Analysis

- First 4 digits of Product ID represents the category.
- Groceries are the most common products sold in Retail Hypermarket.

B) Evaluation

- There is no anomalous products that do not belong to a category.
- There are too many product IDs; high cardinality column.

2.4.6.b Could Product Type be within Product_ID?

Since product category could be found within Product_ID, could the Product_ID also tell us what type of product it is?

In [28]:
```
df['Product_ID'].nunique() # how many product ids are there
```

Out[28]:
```
75725
```

In [29]:
```
# Extract the 5th to 6th digits from the 'Product_ID'
df['Product_Type_1'] = df['Product_ID'].astype(str).str[4:6]

# Extract the 7th to 8th digits from the 'Product_ID'
df['Product_Type_2'] = df['Product_ID'].astype(str).str[6:9]
```

```
In [30]: df['Product_Type_1'].nunique()
```

```
Out[30]: 100
```

```
In [31]: df['Product_Type_2'].nunique()
```

```
Out[31]: 100
```

```
In [32]: # How unique variations of products
         df['Product_Type_3'] = df['Product_ID'].astype(str).str[4:9]
         df['Product_Type_3'].nunique()
```

```
Out[32]: 10000
```

C) Assumptions

1.

- Product_Type_1 could represent the product type; what the product is.
- Product_Type_2 could represent the brand of the brand.

2.

Last 4 numbers could just be random variations to identify a product.

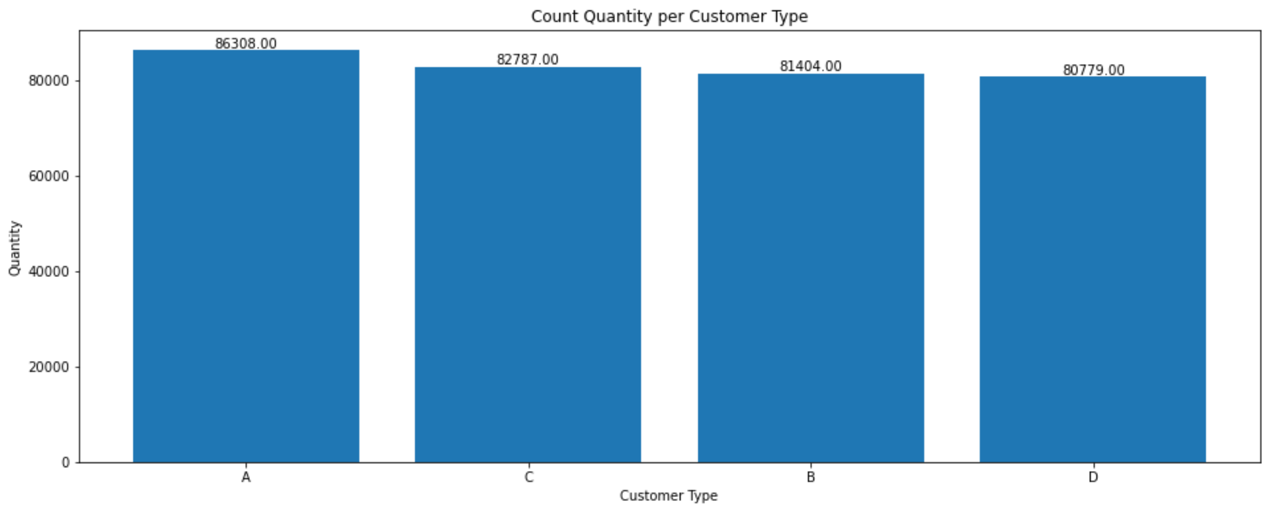
D) Conclusion

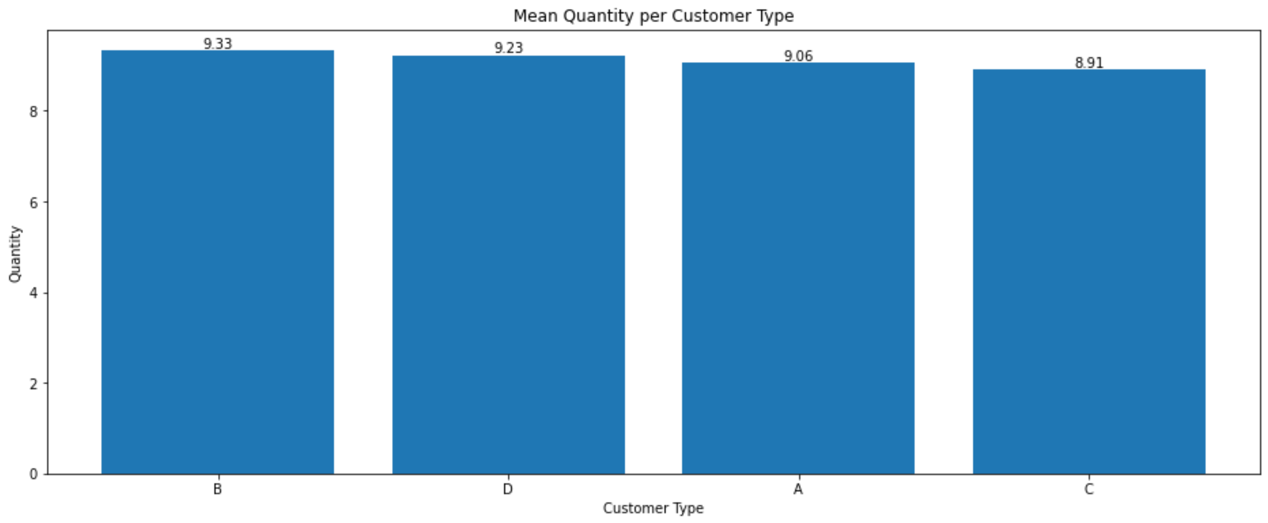
- Do not use Product ID as it could overfit model due to high cardinality.
- **Risky to use Product Types** as its impossible to validate what they represent due to insufficient information. Hence, it could violate the requirements of 'Need to be usable for new products' and it could also potential overfit the model due to high cardinality.

2.4.7 Customer analysis

- I suspect a similar pattern with Customer ID where Retail Hypermarket labels their customer and puts the customer type as the header of the ID. Hence, I investigated each customer class.

In [33]:
```
# Extract the first letter from 'Customer_ID' and create a new column 'Customer_Type'
df['Customer_Type'] = df['Customer_ID'].str[0].str.capitalize()

# how many customer per class
eda_plot(df, 'Customer_Type', 'Quantity', 'bar', 'count')

# average amount of products purchase per transaction for each customer type
eda_plot(df, 'Customer_Type', 'Quantity', 'bar', 'mean')
```





A) Analysis

- First letter of Product ID represents the customer class.
- Around 80K customers for each customer type; no class imbalance in customer type.
- All customer class purchase around 9 products per transaction; indifferent.
- Customer type B has most engagement while customer type C has least engagement.

B) Conclusion

Customer type is a weak feature as there is minimal variation in customer engagement across customer types.

2.5 Multivariate Analysis

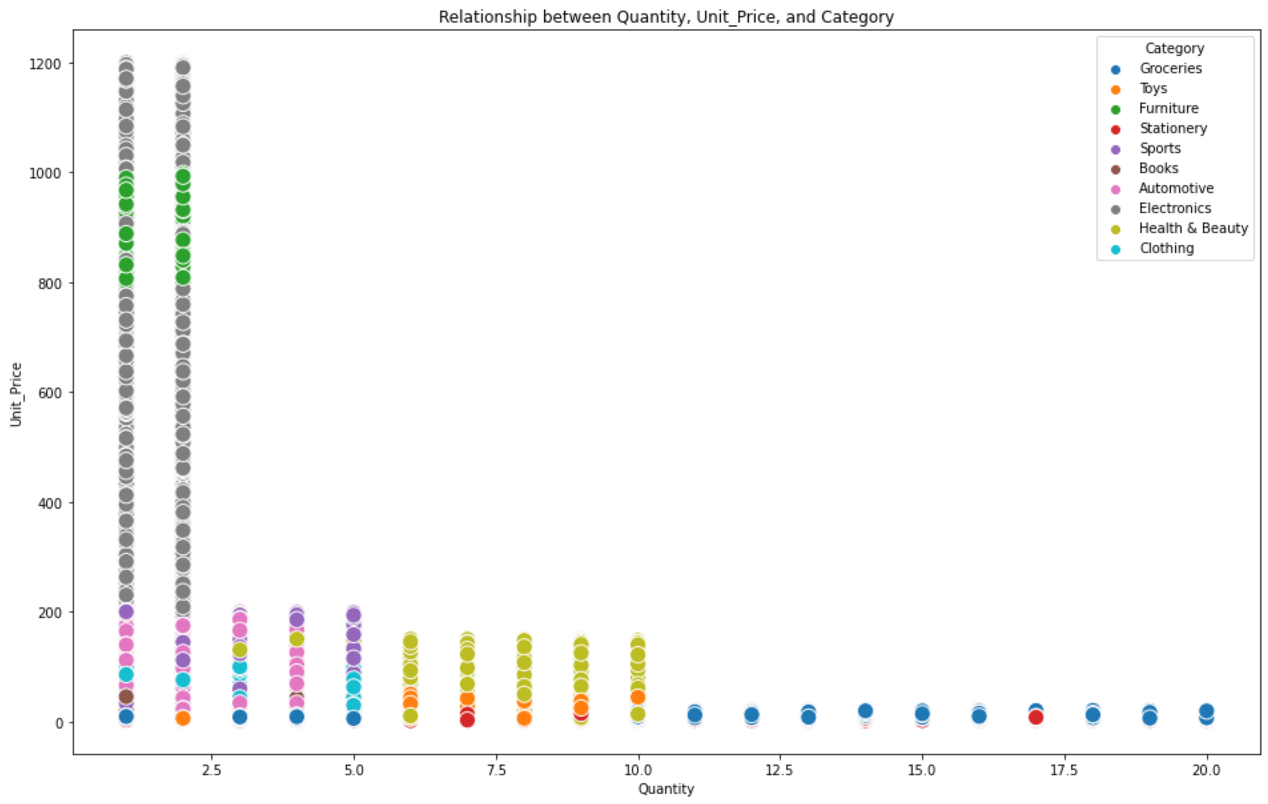
To study 3 features at once, to analyse complex relationships between variables.

In [34]:
```
def multi_analysis(df, x, y, color_by, subset_categories=None):
    dot_size = 150

    if subset_categories:
        df = df[~df[color_by].isin(subset_categories)]

    plt.figure(figsize=(16, 10))
    sns.scatterplot(x=df[x], y=df[y], hue=df[color_by], s=dot_size)
    plt.title(f"Relationship between {x}, {y}, and {color_by}")
    plt.xlabel(x)
    plt.ylabel(y)
    plt.legend(title=color_by)
    plt.show()
```

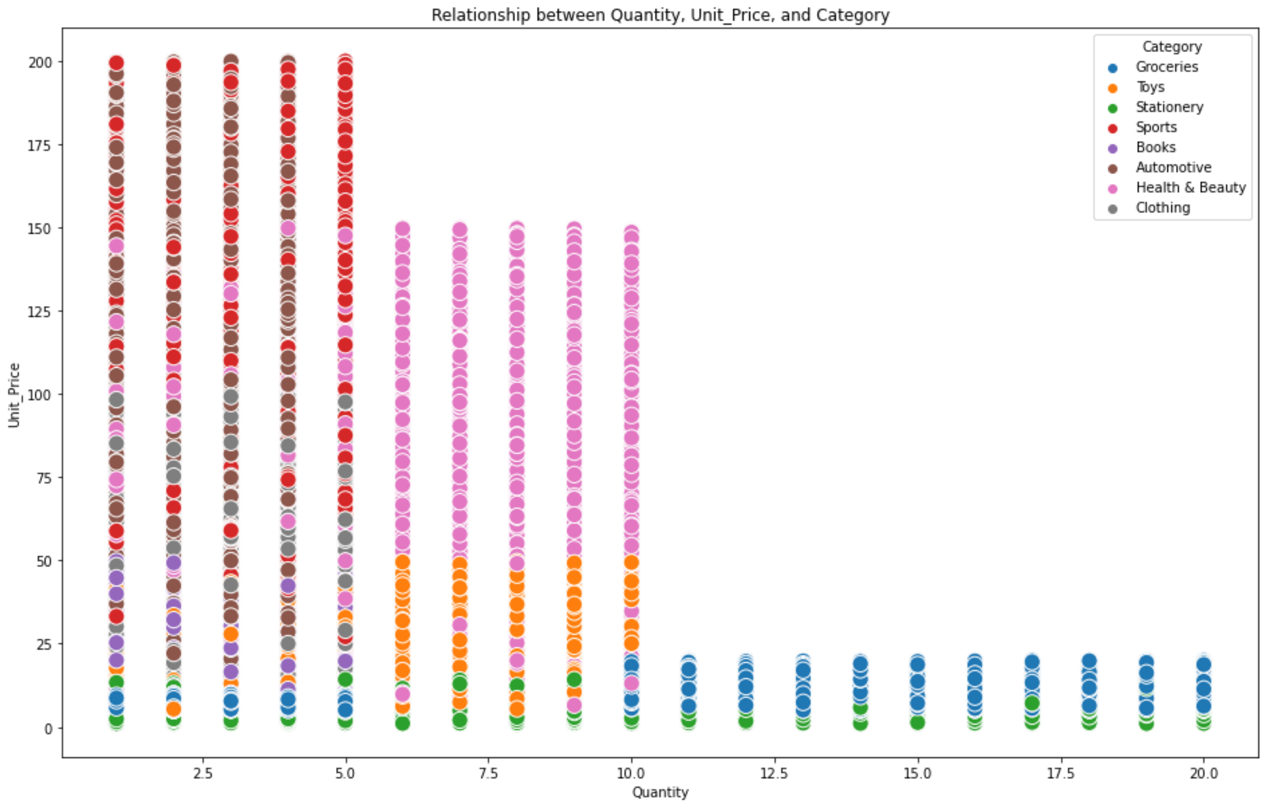
Usage: Input DataFrame, name of x column, name of y column, column that colors the dots, [Unique values in col that colors the dot to exclude].

2.5.1 Follow up for 2.4.5: How does Unit Price AND Category influence the pattern of customer engagement.

In [35]:
```
multi_analysis(df,'Quantity','Unit_Price','Category')
```



```
In [36]: multi_analysis(df,'Quantity','Unit_Price','Category', subset_categories=['Electronics','Furniture']
```



Takeaways:

1. All products above 200 dollars are Electronics or Furnitures, and the maximum engagement these product types received was 2.
2. Furniture products are priced between 800 to 1000 dollars only.
3. Automotive has a maximum possible engagement of 4.
4. Sports, Books, and Clothing products have a maximum possible engagement of 5.
   - Cheaper Books have tendency of higher engagement.
5. Health & Beauty and Toys products have a maximum possible engagement of 10.
6. All Health & Beauty products beyond engagement of 5 are below 150 dollars.
7. Groceries and Stationery products can receive an engagement of up to 20.

Conclusion:

- **Category and Unit Price have overlapping patterns** that could complement each other. For example, unit price can be used to identify certain categories.
- Category is the more useful feature to determine the engagement range for a product category. However, **Unit Price can be used with Category** to narrow down the wide engagement range for categories.

3. Modelling

3.0 Reusable Methods

3.0.1 Importing libraries

In [37]:
```
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score, mean_absolute_error as MAE
from scipy.stats import iqr
from sklearn import tree
import numpy as np
```

3.0.2 Partitioner

To split dataset into train-test sets

Usage:

Provide DataFrame, target column, train size, and whether the predictors should be normalized for interepretability.

In [38]:
```
def partition(X, y, train_size=0.7, SEED=SEED, normalize=True):
    if normalize:
        scaler = StandardScaler()   # init scaler
        X = scaler.fit_transform(X) # scale predictors in X

    # Split the normalized data into train and test sets
    X_train, X_val, y_train, y_val = train_test_split(X, y, train_size=train_size, random_state=SEEI

    return X_train, X_val, y_train, y_val
```

3.0.3 Model Evaluator

Report on model's fit and how predictors explain prediction.

Usage:

Provide the trained model, Predictors DataFrame, Target DataFrame, original Predictors DataFrame (to obtain column names), are you evaluating with model's training dataset? [True/False] default=False, max_depth=[display branches to which level].

In [39]:
```
def evaluate_model(model, X, y, cols, training=False, max_depth=3):
    y_pred = model.predict(X)  # Predict using the model and predictors

    # Only runs for Linear Regression, when intercept_ is found in model
    if hasattr(model, 'intercept_') and training:                                   # ONLY r
        print(f"Intercept: {model.intercept_:.2f}")                                 # print 
        print(pd.DataFrame(model.coef_, cols.columns, columns=["Coefficient"]))     # print 

    # for black-box models like DTR
    elif hasattr(model, 'feature_importances_') and training:
        # Feature Importances
        print("Feature Importances:")
        print(pd.DataFrame(model.feature_importances_, cols.columns, columns=["Importance"]))

        # visualize Tree
        plt.figure(figsize=(16,20))
        tree.plot_tree(model, feature_names=list(X_train.columns), max_depth=max_depth, filled=True

    elif hasattr(model, 'intercept_') or hasattr(model, 'feature_importances_'):
        pass  # if testing set, no need to report on features

    # catch error
    else:
        print("Model type not supported for obtaining coefficients or feature importance.")


    # Visualize the fit
    plt.figure(figsize=(10, 10))
    plt.scatter(y, y_pred, alpha=0.2)
    plt.xlabel("Actual Values")
    plt.ylabel("Predicted Values")
    plt.title("Actual vs Predicted Values")
    plt.grid(True)
    plt.show()

    print(f"R-squared (Goodness of Fit): {r2_score(y, y_pred):.2f}")
```

3.0.4 Accuracy Evaluator

Creates relevant assessments to report on a model's performance.

Usage:

Provide the trained model, Predictors DataFrame, Target DataFrame, sample size for visualization.

```
In [40]: def evaluate_accuracy(model, X, y, sample_size=0):
             y_pred = model.predict(X)  # Predict using the model and predictors

             # accuracy of predictions
             mae = MAE(y, y_pred)
             print(f"Mean Absolute Error (MAE): {mae:.2f}")

             # Visualize Accuracy
             if sample_size > 0:
                 y = y.iloc[:sample_size] # subset y into sample size specified
                 y_pred = y_pred[:sample_size]

             else:
                 return # if no sample size = no need sampling analysis, end the function

             print("Sampled Analysis:")
             plt.figure(figsize=(16, 6))
             plt.plot(y.reset_index(drop=True), "red", label='Actual Data')
             plt.plot(y_pred, 'blue', label='Predicted Data', alpha=0.5)

             # Calculate and plot the 25th and 75th percentiles for y and y_pred
             for data, label, color in [(y, 'Actual Data', 'red'), (y_pred, 'Predicted Data', 'black')]:
                 q1 = np.percentile(data, 25)
                 q3 = np.percentile(data, 75)
                 plt.axhline(q1, color=color, linestyle='dashed', linewidth=2, label=f'{label} 25th percenti
                 plt.axhline(q3, color=color, linestyle='dashed', linewidth=2, label=f'{label} 75th percenti

             plt.ylabel('Quantity')
             plt.title('Actual Vs Predicted')
             plt.legend()
             plt.show()

             # accuracy for subsetted data
             mae = MAE(y, y_pred)
             print(f"MAE for sample: {mae:.2f}")
```

3.0.5 Model Comparer

Compares residuals of models to determine to best performing model based on which model makes more accurate predictions.

Usage:

Provide the trained model1, trained model2, x_val, y_val, sample size for visualization.

In [41]:
```python
def model_comparer(model1, model2, X, y, sample_size=500):
    models = [model1, model2]
    labels = [str(model1), str(model2)]
    colors = ['red', 'teal']

    fig, ax = plt.subplots(figsize=(16, 6))   # Create a figure and axis for the residual plot

    stats = {}                                # Dictionary to store statistics for each model

    for i, (model, label, color) in enumerate(zip(models, labels, colors)):
        y_pred = model.predict(X)             # Make predictions using the current model

        # Limit the dataset to a specified sample size and reset the index to make a smooth residual
        y_sample = y.iloc[:sample_size].reset_index(drop=True)
        y_pred_sample = y_pred[:sample_size]

        residuals = y_sample - y_pred_sample  # Calculate residuals for the current model
        stats[label] = {
            'Q1 (25th percentile)': np.percentile(residuals, 25),
            'Q3 (75th percentile)': np.percentile(residuals, 75)
        }

        # Plot mean and percentiles of the current model
        for stat_name, value in stats[label].items():
            ax.axhline(value, color=color, linestyle='dashed', linewidth=2, label=f'{label} {stat_na

        ax.plot(residuals, color, label=label, alpha=0.6)

    plt.title(f'Residual Plot of {model1} Vs {model2}')
    ax.legend()
    plt.show()

    # Compare IQRs to determine the model with smallest IQR range
    iqr_1 = stats[str(model1)]['Q3 (75th percentile)'] - stats[str(model1)]['Q1 (25th percentile)']
    iqr_2 = stats[str(model2)]['Q3 (75th percentile)'] - stats[str(model2)]['Q1 (25th percentile)']

    if iqr_1 < iqr_2:
        return f"{model1} has the smallest IQR range."
    elif iqr_2 < iqr_1:
        return f"{model2} has the smallest IQR range."
    else:
        return f"Both {model1} and {model2} have the same IQR range."
```

3.1 Data Pre-processing

3.1.0 Encoders

Original columns will be dropped after encoding.

1. One-Hot Encoder:

In [42]:
```python
# One-Hot Encoding
def one_hot_encode(df, columns_to_encode):
    one_hot = pd.get_dummies(df[columns_to_encode])        # Create new col for each unique value
    df = pd.concat([df, one_hot], axis=1)                  # Concatenate encoded columns with the
    df = df.drop(columns=columns_to_encode)                # Drop cols after encoded

    return df
```

Input: DataFrame, [encode_col1, encode_col2]

2.a Label Encoder:

In [43]:
```
from sklearn.preprocessing import LabelEncoder

def label_encode(df, columns_to_encode):
    label_encoders = {}                               # Dictionary to store label encoders fo
    df_encoded = df.copy()                            # Create a copy of the DataFrame to avo

    for col in columns_to_encode:
        label_encoder = LabelEncoder()                    # Init the LabelEncoder
        encoded_data = label_encoder.fit_transform(df_encoded[col])  # Fit and transform the column
        df_encoded[col + '_encoded'] = encoded_data       # Add the encoded column to the

        label_encoders[col] = label_encoder               # Store the label in the dictio
        df_encoded.drop(col, axis=1, inplace=True)        # Drop the original column

    return df_encoded, label_encoders
```

Input: DataFrame, [encode_col1, encode_col2]

2.b Label Decoder:

In [44]:
```
def label_decoder(label_encoders):
    for col, encoder in label_encoders.items():
        print(f"Label values for {col}: {encoder.classes_}") # map out cols and labels
```

3.1.1 Extract relevant features for Model's DataFrame:

In [45]:
```
df.head(1)
```

Out[45]:

	Datetime	Product_ID	Category	Quantity	Unit_Price	Customer_ID	Quarter	Total_Spent	4_digits	Product_Type_1
269220	2022-01-01	20028680	Groceries	15	10.140	c918818917	1	152.100	2002	86

Do not use columns:

1. Datetime:

- Target Leakage: future value.
- High cardinality and leads to overfitting.

2. Product_ID:

- Violates business requirement: Model must work for new unseen values.
- High cardinality and leads to overfitting.

3. Customer ID:

- Violates business requirement: Model must work for new unseen values.
- High cardinality and leads to overfitting.

4. First 4 digits of Product_ID:

- Same meaning and value as category, leading to multicollinearity.

```
In [46]: # Subset predictors and Target into DataFrames for modelling
         X = df[['Category', 'Unit_Price', 'Customer_Type', 'Quarter']] # predictors
         y = df['Quantity']                                              # Target
```

3.1.2 Encoding predictors

I selected one-hot encoding as its easy to interpret and is the most common encoding method used.

```
In [47]: X = one_hot_encode(X, ['Category', 'Customer_Type', 'Quarter'])
```

```
In [48]: X.columns # Check all columns in predictors DataFrame
```

```
Out[48]: Index(['Unit_Price', 'Category_Automotive', 'Category_Books',
                'Category_Clothing', 'Category_Electronics', 'Category_Furniture',
                'Category_Groceries', 'Category_Health & Beauty', 'Category_Sports',
                'Category_Stationery', 'Category_Toys', 'Customer_Type_A',
                'Customer_Type_B', 'Customer_Type_C', 'Customer_Type_D', 'Quarter_1',
                'Quarter_2', 'Quarter_3', 'Quarter_4'],
               dtype='object')
```

3.1.3 Data Partitioning

Splitting dataset into train-test sets where 70% of data is for training and 30% is for validation. Normalization not needed as all features are 0 or 1 except for unit price because their are derived from one-hot encoding. Hence, the model can be interpreted fairly without normalization.

```
In [49]: X_train, X_val, y_train, y_val = partition(X, y, train_size=0.7, normalize=False)
```

3.2 Linear Regression (LR)

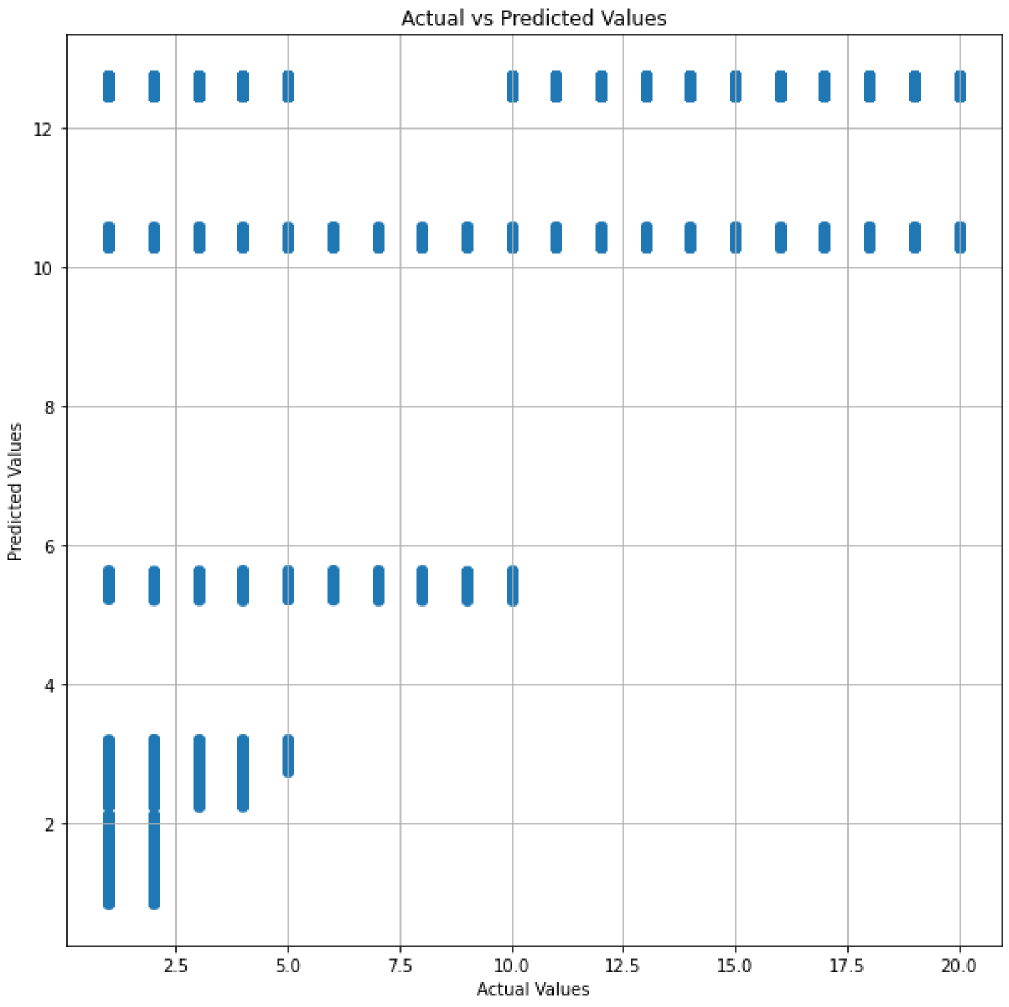
```
In [50]: from sklearn.linear_model import LinearRegression
         LR_model = LinearRegression()       # init model
         LR_model.fit(X_train, y_train)      # train model
```

```
Out[50]: LinearRegression()
```

Evaluate Model explanability on training set:

In [51]:
```
evaluate_model(LR_model, X_train, y_train, X, training=True)
# Model, training predictors, training targets, predictors, isTraining
```

```
Intercept: 4.64
                            Coefficient
Unit_Price                        0.001
Category_Automotive              -2.251
Category_Books                   -1.689
Category_Clothing                -1.712
Category_Electronics             -3.841
Category_Furniture               -4.053
Category_Groceries                7.965
Category_Health & Beauty          0.721
Category_Sports                  -1.754
Category_Stationery               5.795
Category_Toys                     0.820
Customer_Type_A                  -0.001
Customer_Type_B                   0.045
Customer_Type_C                  -0.135
Customer_Type_D                   0.091
Quarter_1                        -0.010
Quarter_2                         0.026
Quarter_3                         0.011
Quarter_4                        -0.026
```



```
R-squared (Goodness of Fit): 0.51
```

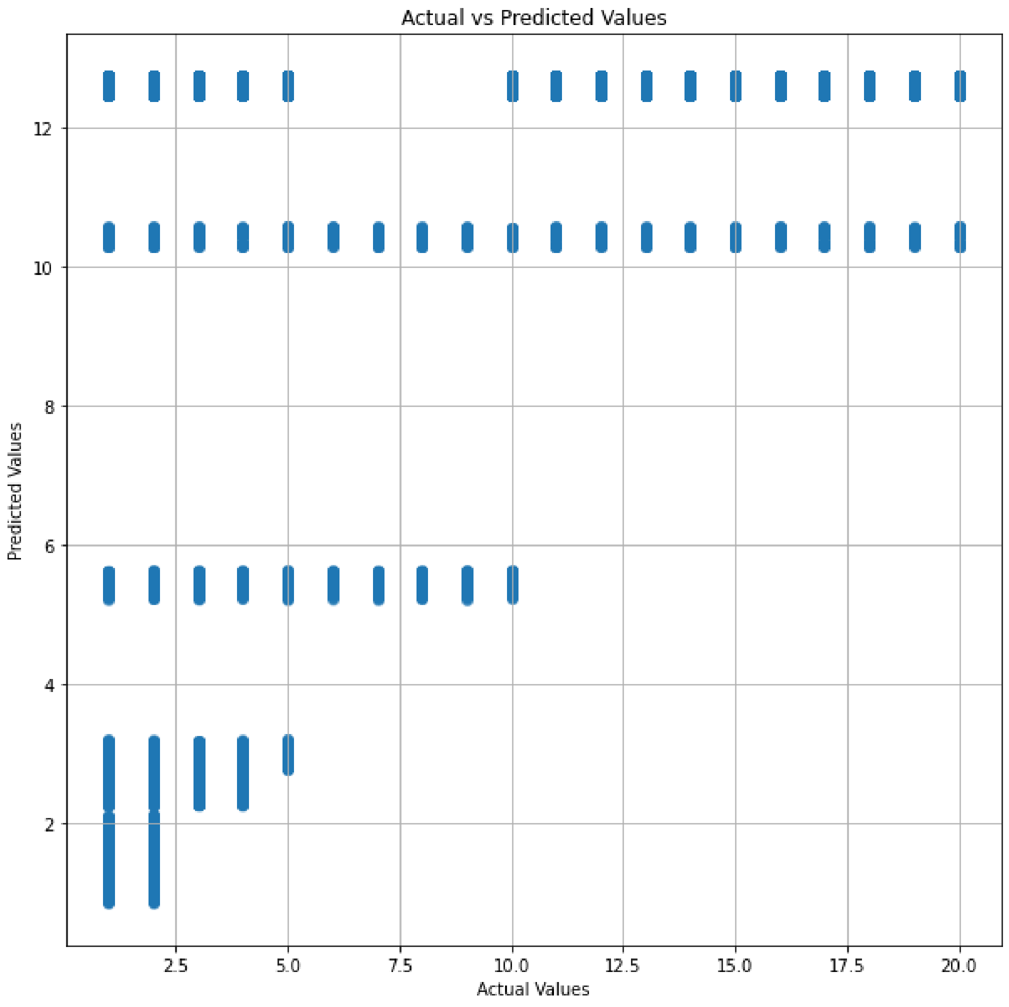
Model performance on training set:

In [52]:
```
evaluate_accuracy(LR_model, X_train, y_train)
```

```
Mean Absolute Error (MAE): 3.36
```

Evaluate Model explanability on validation set:

In [53]:
```
evaluate_model(LR_model, X_val, y_val, X)
```

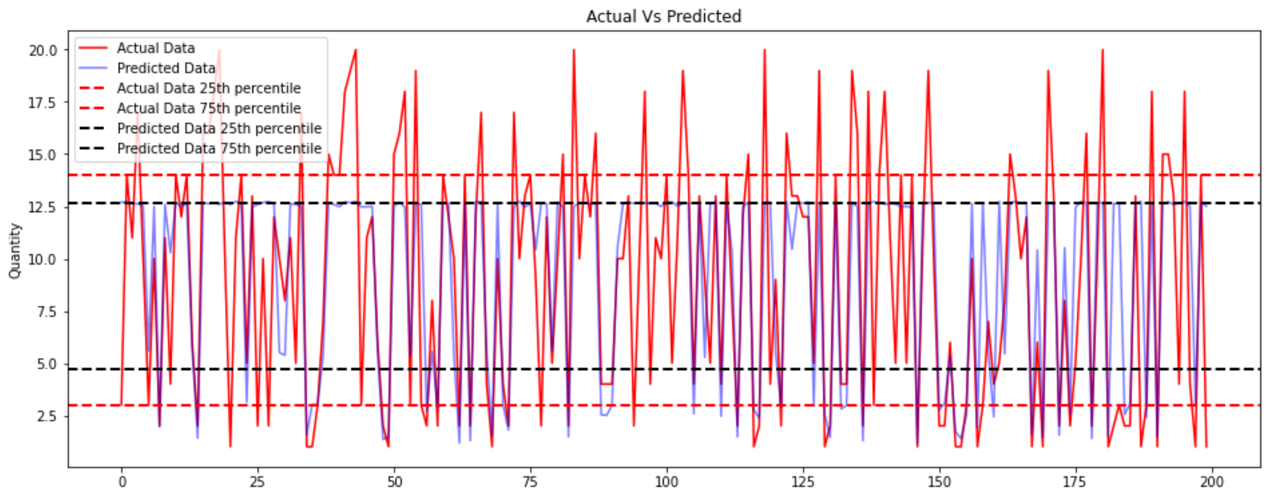


```
R-squared (Goodness of Fit): 0.51
```

Model performance on validation set:

In [54]:
```
evaluate_accuracy(LR_model, X_val, y_val, sample_size=200) # edit sample_size to explore
```

```
Mean Absolute Error (MAE): 3.37
Sampled Analysis:
```



```
MAE for sample: 3.03
```

Analysis:

Model struggles to predict beyond 13 units of Quantity.

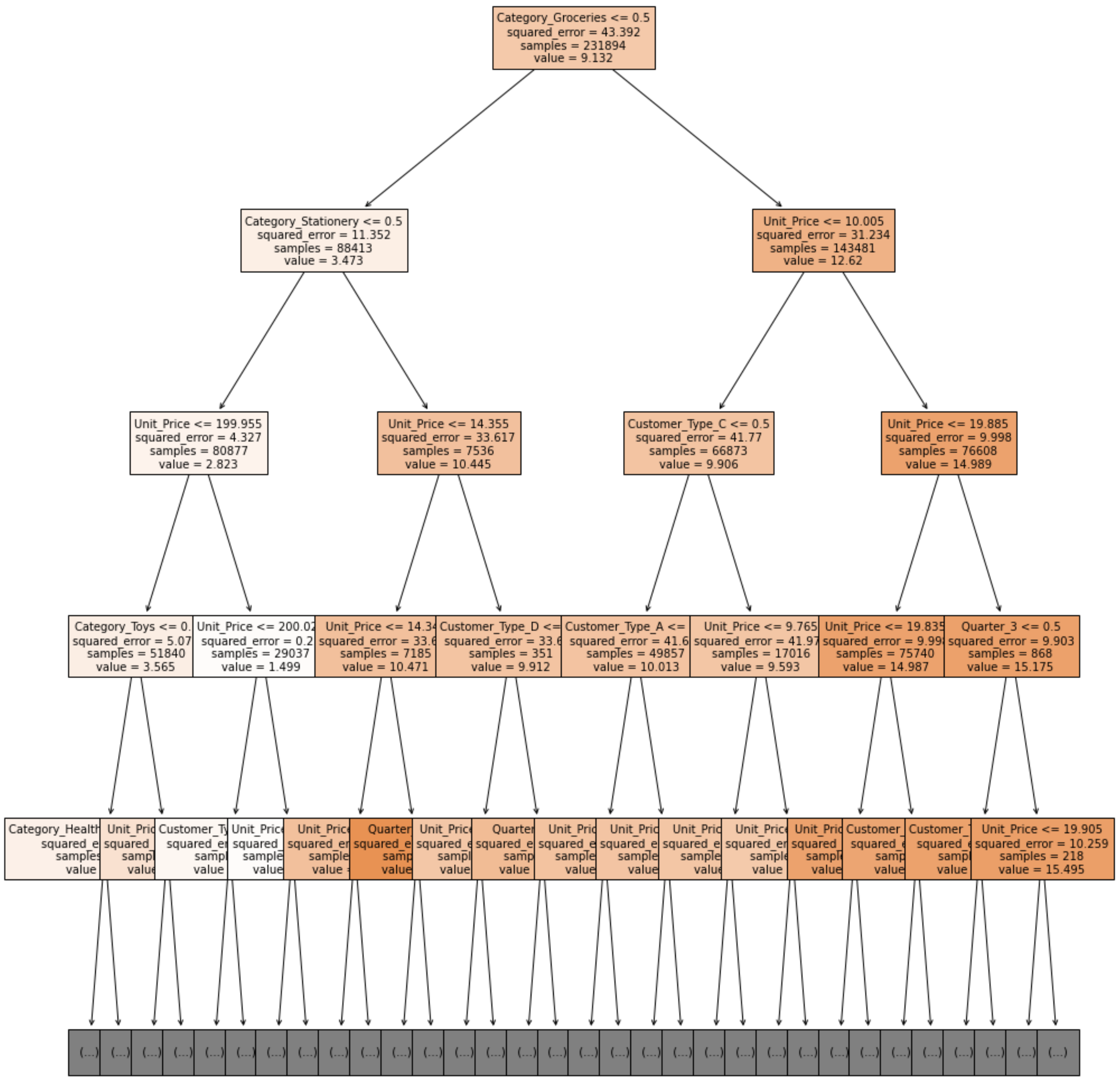
3.3 Decision Tree Regression (DTR)

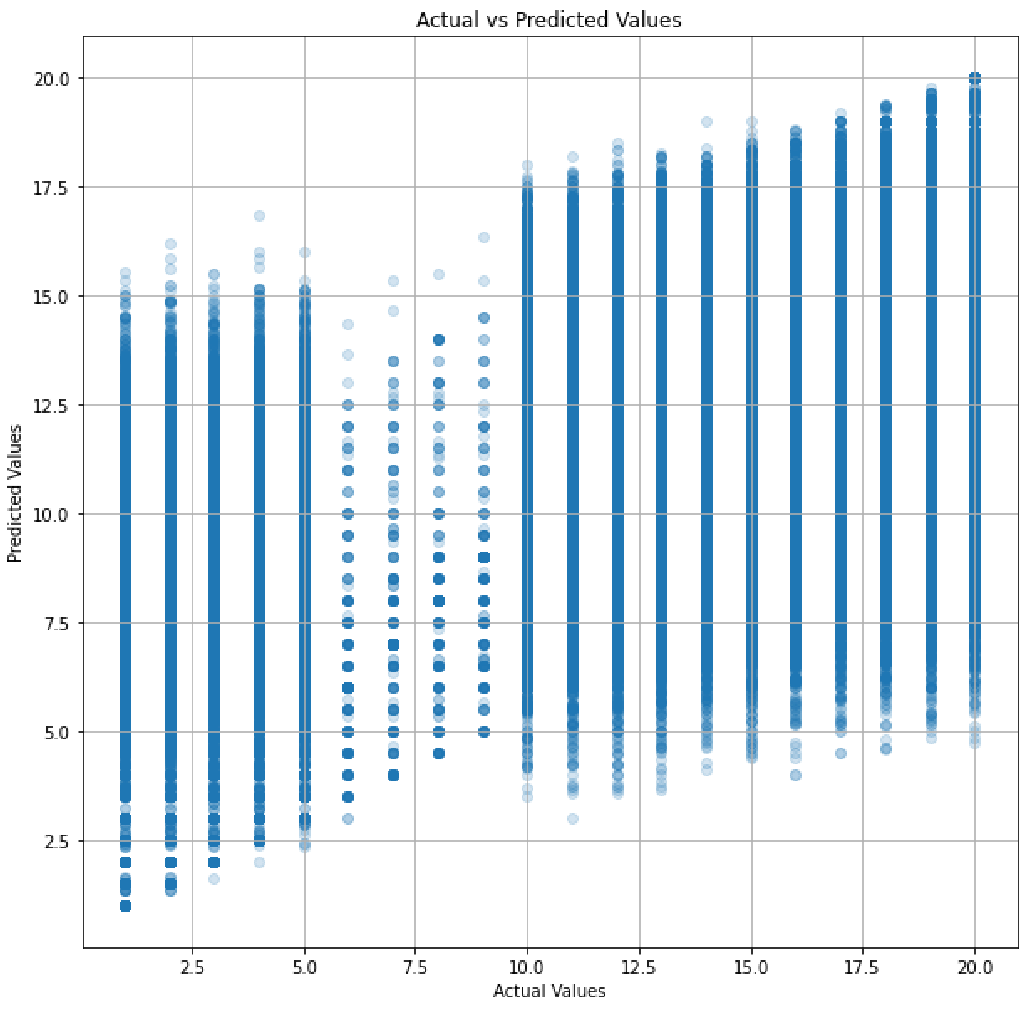
In [55]:
```
#Build DTR model
from sklearn.tree import DecisionTreeRegressor
DTR_model = DecisionTreeRegressor(random_state=SEED).fit(X_train , y_train) # train model
```

Evaluate Model explanability on training set:

In [56]:
```
evaluate_model(DTR_model, X_train, y_train, X, training=True, max_depth=4)
```

```
Feature Importances:
                          Importance
Unit_Price                     0.240
Category_Automotive            0.000
Category_Books                 0.000
Category_Clothing              0.000
Category_Electronics           0.000
Category_Furniture             0.000
Category_Groceries             0.659
Category_Health & Beauty       0.006
Category_Sports                0.000
Category_Stationery            0.058
Category_Toys                  0.005
Customer_Type_A                0.003
Customer_Type_B                0.005
Customer_Type_C                0.003
Customer_Type_D                0.004
Quarter_1                      0.005
Quarter_2                      0.003
Quarter_3                      0.004
Quarter_4                      0.005
```


Category_Groceries <= 0.5
squared_error = 43.392
samples = 231894
value = 9.132
Category_Stationery <= 0.5
squared_error = 11.352
samples = 88413
value = 3.473
Unit_Price <= 10.005
squared_error = 31.234
samples = 143481
value = 12.62
Unit_Price <= 199.955
squared_error = 4.327
samples = 80877
value = 2.823
Unit_Price <= 14.355
squared_error = 33.617
samples = 7536
value = 10.445
Customer_Type_C <= 0.5
squared_error = 41.77
samples = 66873
value = 9.906
Unit_Price <= 19.885
squared_error = 9.998
samples = 76608
value = 14.989
Category_Toys <= 0.
squared_error = 5.07
samples = 51840
value = 3.565
Unit_Price <= 200.02
squared_error = 0.2
samples = 29037
value = 1.499
Unit_Price <= 14.34
squared_error = 33.6
samples = 7185
value = 10.471
Customer_Type_D <=
squared_error = 33.6
samples = 351
value = 9.912
Customer_Type_A <=
squared_error = 41.6
samples = 49857
value = 10.013
Unit_Price <= 9.765
squared_error = 41.97
samples = 17016
value = 9.593
Unit_Price <= 19.835
squared_error = 9.99
samples = 75740
value = 14.987
Quarter_3 <= 0.5
squared_error = 9.903
samples = 868
value = 15.175
Unit_Price <= 19.905
squared_error = 10.259
samples = 218
value = 15.495
(...)



```
R-squared (Goodness of Fit): 0.69
```

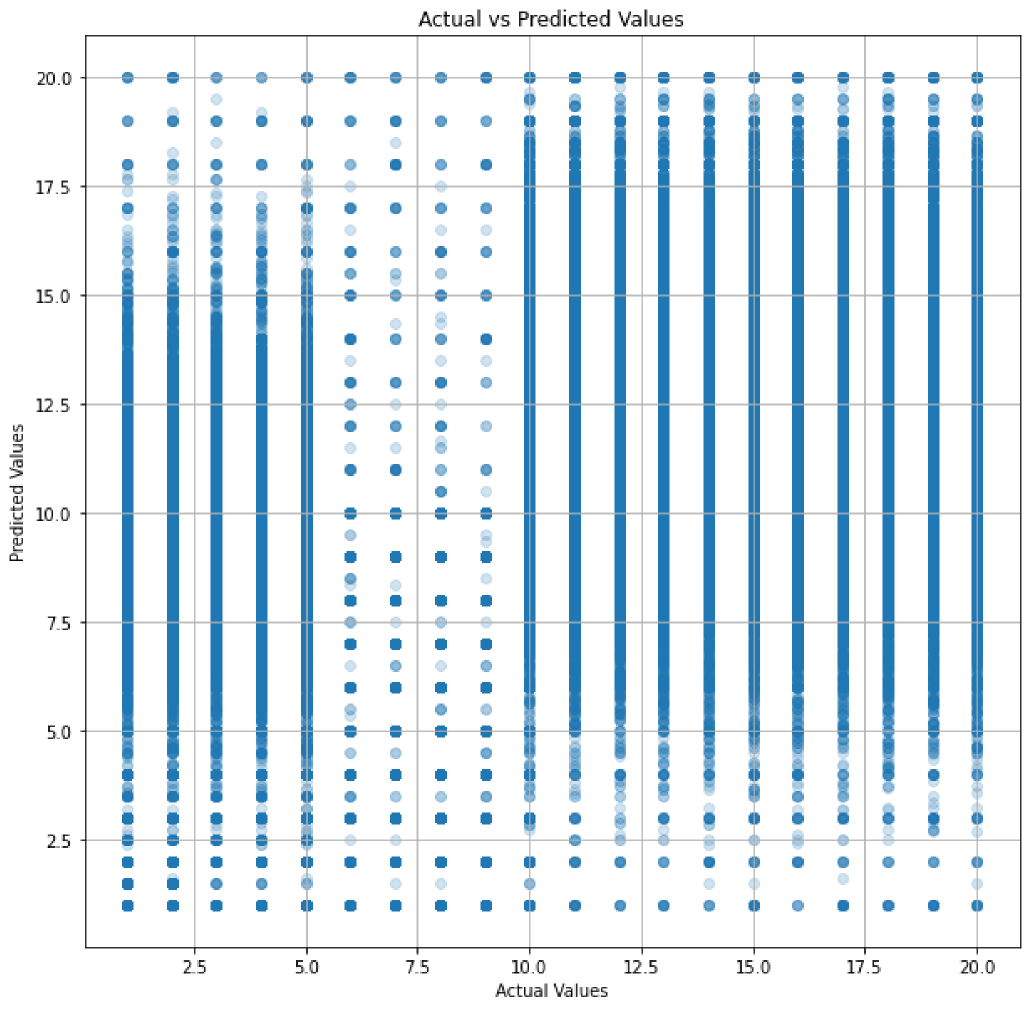
Model performance on training set:

```
In [57]: evaluate_accuracy(DTR_model, X_train, y_train)
```

```
Mean Absolute Error (MAE): 2.36
```

Evaluate Model explanability on validation set:

```
In [58]: evaluate_model(DTR_model, X_val, y_val, X)
```

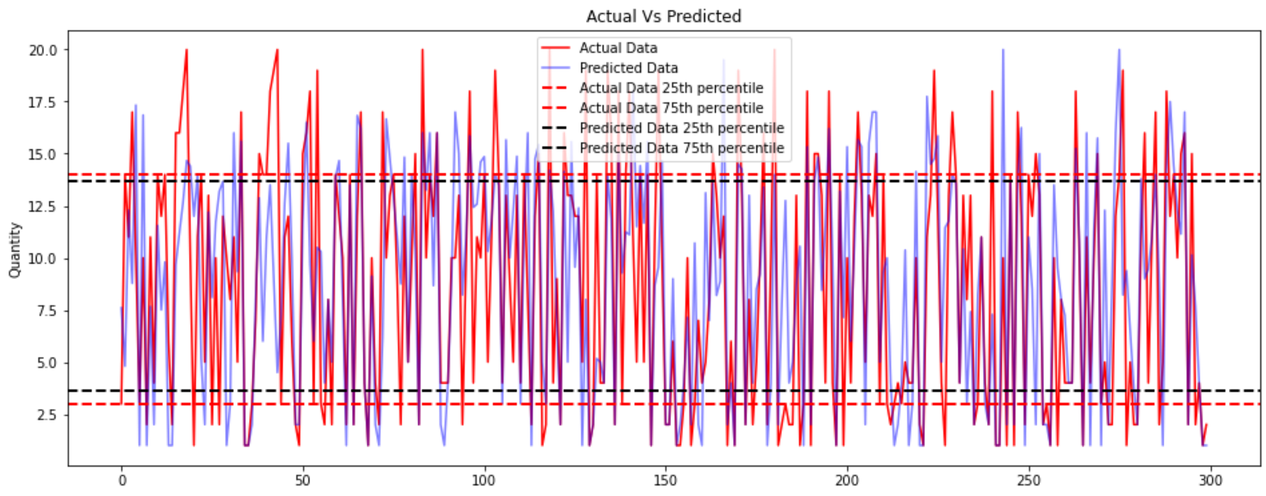


```
R-squared (Goodness of Fit): 0.50
```

Model performance on validation set:

```
In [59]: evaluate_accuracy(DTR_model, X_val, y_val, sample_size=300) # edit sample_size to explore
```

```
Mean Absolute Error (MAE): 3.46
Sampled Analysis:
```



```
MAE for sample: 3.30
```

Analysis:

DTR model is likely overfitted due to disparity of the R-Squared and MAE for training and validation.

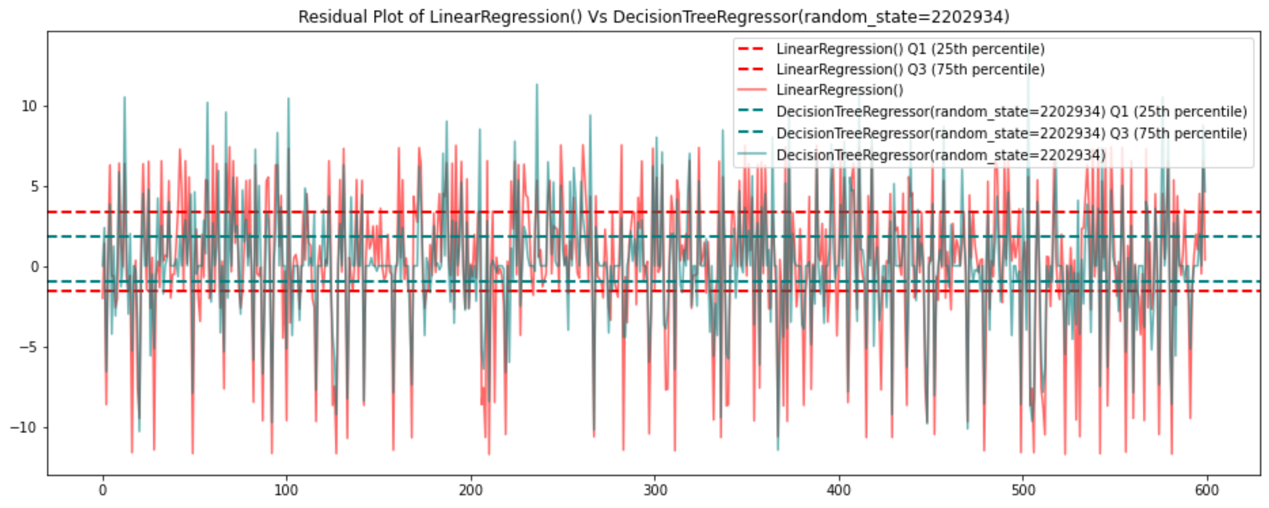
Evaluation of Baseline Models:

- LR is struggling to capture customer engagement beyond 13 units.
- DTR has overfitting issues that needs to be fixed.
- DTR seems to have potential in understanding customer engagement as the training MAE is 2.36. Improvements should be made to make it lower.
- DTR's R-Squared is below 80%, indicating that the model has limited explanatory power as the patterns in the variations of the predictors were not captured adequately.
- I need to round off prediction to whole value or tune to model to only produce discrete predictions.

4. Model comparison

4.1 Training Dataset

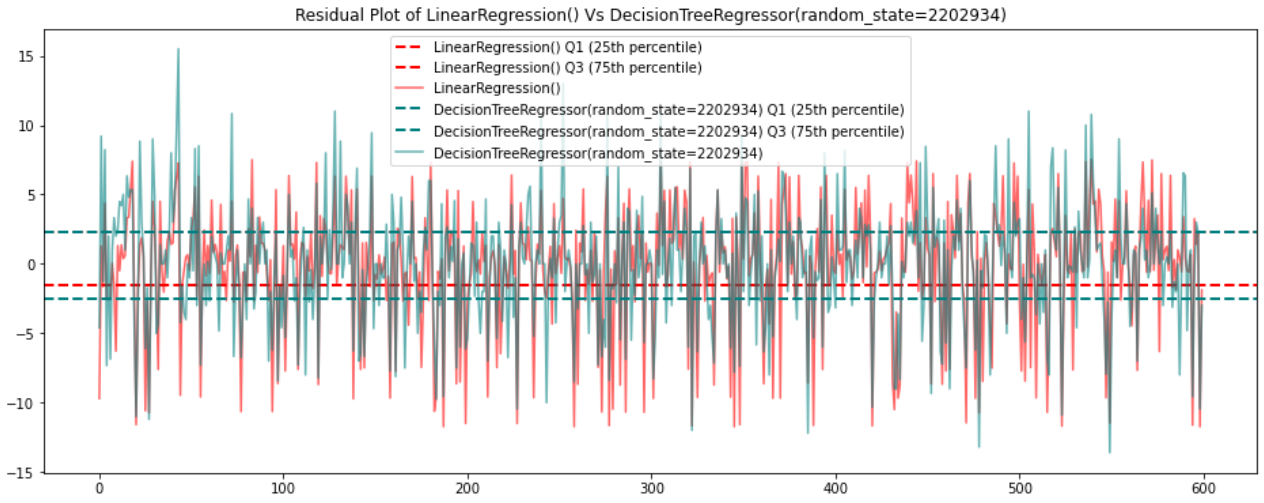
```
In [60]: model_comparer(LR_model, DTR_model, X_train, y_train, sample_size=600)
```



```
Out[60]: 'DecisionTreeRegressor(random_state=2202934) has the smallest IQR range.'
```

4.2 Validation Dataset

In [61]:
```
model_comparer(LR_model, DTR_model, X_val, y_val, sample_size=600)
```



Out[61]:
```
'LinearRegression() has the smallest IQR range.'
```

Next Steps for Part 2

Build upon the DTR model by:

- Addressing overfitting issue
- Features Engineering for more meaningful predictors
- Features Selection
- Trying out Ensemble Machine Learning methods like Random Forest
- Selecting best Encoding Method
- Permutation Selection
- Model hyperparameter-tuning/pruning to improve predictive performance

References:

- 1.0 Problem Framing: Creating Week 03 Workshop 05: Problem Framing. Temasek Polytechnic.
- 1.6 Metrics for evaluating models. Week 02 Workshop 04: Model Scoring
- 1.8 Considering environmental factors: Ideas were improved upon suggestions from ChatGPT.
- 1.9 Formulating potential Target Leakages, Week 03 Workshop 06: Target Leakage
- 2.1 Data Dictionary is taken from project specifications.
- 2.4.2 EDA Graph Plotter is built with ChatGPT. OpenAI. (2023, October 30). Re: Python Code for Creating Line and Bar Plots in EDA [Online Forum Comment]. ChatGPT by OpenAI. https://www.chatgpt.com (https://www.chatgpt.com)
- 2.4.6 Graph of Total Revenue per Product is adapted from ChatGPT's code. OpenAI. (2023, October 30). Re: Python Code for Creating Line and Bar Plots in EDA [Online Forum Comment]. ChatGPT by OpenAI. https://www.chatgpt.com (https://www.chatgpt.com)
- 3.0 Reusable methods are built with ChatGPT. OpenAI. (2023, October 31).
- 3.0.3 Coefficient of predictors table adapted from P01_RecapML 4.4 Model Interpretation#Coefficients. Temasek Polytechnic.
- 3.0.4 Sampled analysis graph is adapted from P01_RecapML 4.4 Model Interpretation#Plot of y_pred and y_test. Temasek Polytechnic.
- 3.0.5 Model comparer built with Lab P02 and ChatGPT. OpenAI. (2023). 3.0.5 Model Comparer [Source code]. OpenAI GPT-3.5. https://chat.openai.com/share/5634ce5f-5e03-4e4c-ac90-2dca9cd206e6 (https://chat.openai.com/share/5634ce5f-5e03-4e4c-ac90-2dca9cd206e6)
- 3.1.0 One-Hot encoder built with ChatGPT. OpenAI. (2023, October 31). Re: # One-Hot Encoding def one-hot(): // create a one-hot encoder function. input: df, columns to be encoder [Online Forum Comment]. ChatGPT by OpenAI. https://www.chatgpt.com (https://www.chatgpt.com)
- 3.1.0 Label encoder and decoder built using One-Hot encoder and Lab P01_RecapML#3. Data Preparation

- 3.3 Code to visualize Decision Tree Regressor is taken from Lab P02_TreeBasedAlgorithm#2.4 Limitations of DT#Method 2

Guide for Data Modelling:

- Ameisen, E. (2018, March 6). Always start with a stupid model, no exceptions. Medium. https://blog.insightdatascience.com/always-start-with-a-stupid-model-no-exceptions-3a22314b9aaa (https://blog.insightdatascience.com/always-start-with-a-stupid-model-no-exceptions-3a22314b9aaa)
- Nair, A. (2022, April 9). Baseline Models: Your Guide For Model Building - Towards Data Science. Medium. https://towardsdatascience.com/baseline-models-your-guide-for-model-building-1ec3aa244b8d

End of Part 1