Designing and Implementing a Data Warehouse

By Abdulkadir Abdulkadir

QA

Abstract

This report demonstrates how a data warehouse for business intelligence. The project starts off with an initial case study of a fast food chain known as Galleria, provided with a data set and from there. With these two, a comparative analysis between an on-premise, cloud and hybrid solution is evaluated. Additionally, the data set provided is then cleaned and understood using exploratory data analysis using python on Jupyter Notebook. After that, an OLAP star schema is designed on SQL, with a source to target map demonstrating how the data has been mapped out. Then, a test is used to verify the success of an OLAR star schema and finally, a dashboard is crated on PowerBI as a means of a reporting tool for business intelligence to answer the companies questions.

Contents

1 Introduction **1**
1.1 Case Study 1
1.2 Project Outline 1

2 Problem Domain Understanding **2**
2.1 On-Premise vs Cloud vs Hybrid 2
2.2 Business Requirements 3

3 Data Understanding **4**
3.1 Understanding and Cleaning the Data 4
3.2 Exploratory Data Analysis 10
3.2.1 Numerical Data 10
3.2.2 Categorical Data 12

4 OLAP Schema Design **16**
4.1 Looking at the Data 16
4.2 Design of Star Schema 17

5 Source to Target Mapping **24**

6 Test Approach **24**

7 Visualisations **26**

1 Introduction

1.1 Case Study

Galleria Holdings is a fast-food chain which originated in Italy but has now acquired a number of businesses in the UK. The company believes that the UK menus are too large and inconsistent across the outlets they have acquired. They also believe that the menus should be consolidated to suit the preferences of the customers and therefore reduce unnecessary purchases of food which is not chosen by customers.

In order to achieve this, the parent company wishes to create a data warehouse for Business Intelligence (BI) purposes.

They wish to analyse sales to determine the most popular menu items and also those which produce the most revenue. They also wish to find out the most popular product groups and those which provide the most revenue. Finally, they wish to construct a "league table" of the total weekly and monthly sales of the various outlets so that their sales performance can be monitored easily.

1.2 Project Outline

In this project, a data warehouse is to be designed and created whereby BI solutions can be created to answer Galleria's business questions and provide for their needs. What is provided in this project is simply a data set which needs to be transformed into a data warehouse using a schema which will then have to be turned into visualisations for analysis of specific business questions. Consequently, the project has been split up into six sections and they are as follows:

- **Section 2:** The problem domain understanding which consists of conducting a comparative analysis of on-premise, cloud-based and hybrid data engineering solutions and identifying the best choice for this case. Additionally, this section also reviews the business requirements and identifies any additional relevant business questions for reporting purposes.
- **Section 3:** The understanding of the data which consists of looking at the data presented and providing an initial assessment of the data by understanding what is in the data and how each variable is related to each other. It also looks at the data quality issues and prepares the data for the creation of the warehouse.
- **Section 4:** The OLAP schema design which consists of identifying which schema will be implemented and why that is the case as well as demonstrating the design.
- **Section 5:** Source to target mapping which shows how the data will be mapped into different tables and they all connect together.
- **Section 6:** Test approach which shows the validation of the splitting of the data into an OLAP schema and checking whether it has been done successfully.
- **Section 7:** Visualisation which will show the final analytical solutions to the questions asked and more.

2 Problem Domain Understanding

When thinking about implementing a data warehouse solution for Galleria, it is important to understand the differences between an on-premise solution to a cloud or hybrid solution. This section will consider each case.

2.1 On-Premise vs Cloud vs Hybrid

A comparative analysis between on-premise, cloud and hybrid will be discussed for eight different evaluation criteria; security, compliance, scalability, efficiency, reliability, fidelity, flexibility, portability. Table 1 summarises the differences of these criteria.

Table 1: Comparative analysis of on-premise, cloud based and hybrid data engineering solutions.

	On-Premise	Cloud	Hyrbrid
Security	Security is the sole responsibility of the company. Having full control over data and services.	Shared responsibility model. The vendor manages the security of the infrastructure. The company is responsible for the security of the data and services.	Combination of Public and Private.
Compliance	There are regulatory controls that most companies need to abide by. To meet these government and industry regulations, it is imperative that companies remain complaint and have their data in place. This can easily be done if all the data is maintained in-house.	Companies need to ensure that the service provider is meeting the regulatory mandates within their specific industry. It is important that the data of customers, employees and partners is secure, whereby ensuring privacy.	Combination of Public and Private
Scalability	Not easily scalable. The company has to pay to deploy new servers. It is also time consuming.	Easily scalable. Can scale up or down and the company only pays for what they use. Pay as you go scheme.	Can scale up on cloud whilst maintaining on-premise
Efficiency	Maintaining infrastructure costs working time i.e. operational expenditure as well as initial capital expenditure. Must maintain software and hardware. Not easy to scale up. Is not dependent on the internet.	Only pay for the resources you need which minimises cost i.e.operational costs. No capital expenditure/ Can access remotely. Easy to scale. Easy data recovery. No capital expenditure.. Software and hardware maintained.	Has the positive and negatives of both models.
Reliability	If the data centre goes down, it will take time to get it running again which could cost the company a lot of money.	Very reliable uptime due to multiple data centre locations and availability zones.	Has the negative of on-premise.
Fidelity	Not easily reproducible and not as quick to reproduce as cloud.	Easily reproducible. Can save and replicate templates of resources and subscriptions.	Has the negative of on-premise.
Flexibility	Up front costs can be expensive and it costs to maintain the infrastructure. It may be expensive to upgrade. Can only be accessed in its location. On-premise ERP systems can be accessed remotely but often requires third party support which results in risk of security and communication failures.	Can easily adapt infrastructure to needs without making large investments. Can access resources from anywhere. All that is required is an internet connection.	Has the flexibility of cloud.
Portability	Easily portable to the cloud with pay as you go plan. Difficult to move a data centre from one location to another.	Can move from one cloud vendor to another or from cloud vendor to on-premise. However, it costs to move from the cloud vendor.	Has the negative of on-premise.

Looking at the data set provided to understand the magnitude of Galleria can play an important role when deciding if they should run on-premise or cloud.

Galleria is a relatively small company in the UK as it has had a total of almost 17,000 customers, only almost 50,000 rows of data, it is a company that has more room for growth and a company that has been growing. From 2016 with a total of 1334 customers, to 2017 with 3279, to 2018 with 6296, and finally 2019 with 5825 customers, Galleria is a growing company and hence, can benefit more by choosing to run their business through a cloud service. That is because as a company that is growing each year, obtaining more and more customers, scalability is very important. Galleria should be able to easily scale up their business needs to match the demand of customers and that is easily attainable using cloud than on-premise.

Additionally, Galleria do not have to pay up capital expenditure to build a data centre to run their business but rather can use the pay as you go method on the cloud to only pay for what they need and save money and time. The cloud is a very efficient system that can help the company save time which they can spend looking to grow the business. Hence, they should opt to cloud services rather than on-premise.

2.2 Business Requirements

The business requirements for Galleria is that they wish to analyse the sales data to determine the most popular menu items and also those which produce the most revenue. With this information, they can consolidate their menu and provide items which the customers prefer. They also wish to find out the most popular product groups and those which provide the most revenue. Finally, they wish to construct a league table of the total weekly and monthly sales of the various outlets so that their sales performance can be monitored easily.

Additional sales analysis that Galleria can use to understand their sales patterns are looking at how their revenue is affected by the time of the year as well as how it has varied throughout the years. Moreover, a useful analysis would also be to look at which outlet is performing the best in sales and the most popular items in each outlet so that they can cater specifically for those outlets to maximise sales. This can solve their issue of inconsistent menus from outlet to outlet and cater to the preferences of their customers.

3 Data Understanding

This section is an initial assessment of the data presented by Galleria fast food chain and obtaining a complete understanding of what is in the data in order to prepare for the creation of a data warehouse. By using Jupyter notebook, a comprehensive understanding can be obtained.

3.1 Understanding and Cleaning the Data

Firstly, it is important to have a quick glance at the data at hand and that is demonstrated in figure 1.

```
df.head()
```

	SaleDate	TicketNo	Outlet	Total	OrderQty	Stock_Code	Name	Description	Price	Product_Group	Group_name	CardType
0	2018-08-21	92208	Birmingham	5.0	1	BK0101	Eggs Florentine	NaN	5.0	B	Breakfast	Visa
1	2018-08-21	92209	Birmingham	7.5	1	CP0125	The Sicilian Crepe	Tuna, black olives, cheddar cheese, spinach an...	7.5	C	Crepes and Galettes	Avis
2	2018-08-21	92211	Middlesborough	7.5	1	CP0170	Crepe Japonnais	Teriyaky Salmon & Shitake Mushrooms, with salad	7.5	C	Crepes and Galettes	Visa
3	2018-08-21	92217	Birmingham	5.0	1	BK0101	Eggs Florentine	NaN	5.0	B	Breakfast	Avis
4	2018-08-21	92217	Birmingham	7.5	1	CP0125	The Sicilian Crepe	Tuna, black olives, cheddar cheese, spinach an...	7.5	C	Crepes and Galettes	Avis

Figure 1: First five rows of the data.

The next step is to understand the data types of each variable, as shown in figure 2.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 40705 entries, 0 to 40704
Data columns (total 12 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   SaleDate       40705 non-null  object
 1   TicketNo       40705 non-null  int64
 2   Outlet         40705 non-null  object
 3   Total          40705 non-null  float64
 4   OrderQty       40705 non-null  int64
 5   Stock_Code     40705 non-null  object
 6   Name           40705 non-null  object
 7   Description    29754 non-null  object
 8   Price          40705 non-null  float64
 9   Product_Group  40705 non-null  object
 10  Group_name     40705 non-null  object
 11  CardType       40705 non-null  object
dtypes: float64(2), int64(2), object(8)
memory usage: 3.7+ MB
```

Figure 2: Variable data types.

Another way of understanding the data you have is to look at the unique values in the data set as shown in figure 3.

```python
# look at unique values each variable has
for col_name in df.columns:
    print(col_name, end=': ')
    print(df[col_name].value_counts().count())
```

```
SaleDate: 1123
TicketNo: 16708
Outlet: 10
Total: 204
OrderQty: 28
Stock_Code: 78
Name: 77
Description: 64
Price: 28
Product_Group: 9
Group_name: 9
CardType: 4
```

Figure 3: Unique variables.

For cleanliness of data, the first approach is to look for null values as show in figure 4.

```python
df.isnull().sum()
```

```
SaleDate             0
TicketNo             0
Outlet               0
Total                0
OrderQty             0
Stock_Code           0
Name                 0
Description      10951
Price                0
Product_Group        0
Group_name           0
CardType             0
dtype: int64
```

The NaN values for description is not an issue as it is just additional data explaining what the item is

Figure 4: Null values.

Additional measures can be taken to verify cleanliness of data, for example, looking at a sample of the data and seeing if it is clean as shown in figure 5.

```
1 df.iloc[11]
```

```
SaleDate                                            2018-08-21
TicketNo                                                 92235
Outlet                                           Middlesborough
Total                                                      9.5
OrderQty                                                     1
Stock_Code                                              PZ0005
Name                                                CAPRICCIOSA
Description      Tomato base, Mozzarella, ham & mushrooms
Price                                                      9.5
Product_Group                                            P
Group_name                                               Pizza
CardType                                            Debit Card
Name: 11, dtype: object
```

```
1 df.iloc[15]
```

```
SaleDate                                            2018-08-22
TicketNo                                                 92247
Outlet                                             Peterborough
Total                                                      7.5
OrderQty                                                     1
Stock_Code                                              CP0170
Name                                             Crepe Japonnais
Description      Teriyaky Salmon & Shitake Mushrooms, with salad
Price                                                      7.5
Product_Group                                                C
Group_name                                   Crepes and Galettes
CardType                                            Debit Card
Name: 15, dtype: object
```

```
1 df['Description'].iloc[15]
```

```
'Teriyaky Salmon & Shitake Mushrooms, with salad'
```

```
1 df['Description'].iloc[13]
```

```
nan
```

```
1 df['Description'].iloc[0]
```

```
nan
```

```
1 df['Description'].iloc[11]
```

```
'Tomato base, Mozzarella, ham & mushrooms '
```

```
1 df['Product_Group'].iloc[11]
```

```
'P    '
```

There are a lot of cells with extra white space that needs to be removed and potentially find cells which may just be spaces and no actual data

Figure 5: Sample data.

As white space is present in the data, it is sensible to check whether it is occurring in the rest of the variables, as there is not unique values anyways.

Look at object columns for whitespace as floats and integers cannot take spaces

```python
for col_name in df:
    try:
        unique_vals = np.unique(df[col_name])
        nr_vals = len(unique_vals)
    except:
        unique_vals = np.unique(df[col_name].astype(str))
        nr_vals = len(unique_vals)

    print(f"Unique values in column '{col_name}': {nr_vals}")
    print(unique_vals, end='\n\n')
```

```
Unique values in column 'SaleDate': 1123
['2016-07-01' '2016-07-02' '2016-07-03' ... '2019-07-29' '2019-07-30'
 '2019-07-31']

Unique values in column 'TicketNo': 16708
[ 82191  82192  82193 ... 114778 114780 114781]

Unique values in column 'Outlet': 10
['Birmingham' 'Cardiff' 'Edinburgh' 'Ipswich' 'London' 'Middlesborough'
 'Peterborough' 'Poole' 'Weymouth' 'Worthing']

Unique values in column 'Total': 204
[  3.     3.25   3.5    3.75   4.     4.5    5.     5.5    5.75   6.
   6.5    6.75   7.     7.5    8.     8.5    9.     9.5    9.75  10.
  10.5   11.    11.25  11.5   11.75  12.    12.5   13.    13.5   14.
  14.5   15.    16.    16.25  16.5   17.    17.25  17.5   18.    18.75
  19.    19.5   20.    20.25  21.    22.    22.5   22.75  23.    23.5
  24.    24.5   25.    25.5   26.    26.25  27.    27.5   28.    28.5
  28.75  29.    29.25  30.    31.5   32.    32.5   33.    33.75  34.
  34.5   35.    35.25  35.75  36.    37.5   38.    38.5   39.    40.
  40.25  40.5   41.25  42.    42.25  42.5   43.5   44.    45.    45.5
  46.    47.    47.25  47.5   48.    48.75  49.    50.    51.    52.
  52.5   54.    55.    55.25  56.    57.    57.5   58.    58.5   58.75
  59.5   60.    60.75  62.5   63.    64.    65.    66.    66.5   67.5
  68.    69.    70.    70.5   71.5   72.    72.5   73.5   74.25  75.
  76.    76.5   77.    78.    78.75  80.    80.5   81.    82.25  82.5
  84.    84.5   85.    85.5   87.5   87.75  88.    90.    91.    92.
  94.    94.5   95.    96.    97.5   98.   100.   101.25 103.5  104.
 104.5  105.   108.   110.   112.   112.5  114.   114.75 115.   117.5
 119.   120.   121.   121.5  123.5  125.   126.   126.5  127.5  128.
 128.25 129.25 130.   133.   135.   136.5  140.   141.   141.75 142.5
 147.   148.5  150.   152.   155.25 157.5  160.   161.   161.5  172.5
 180.   187.5  202.5  216.  ]

Unique values in column 'OrderQty': 28
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
 25 26 27 32]

Unique values in column 'Stock_Code': 78
['APP001' 'APP002' 'BK0101' 'BK0102' 'BK0103' 'BK0110' 'BK0160' 'BK0161'
 'BK0162' 'CP0101' 'CP0102' 'CP0105' 'CP0107' 'CP0108' 'CP0110' 'CP0111'
 'CP0112' 'CP0113' 'CP0117' 'CP0118' 'CP0119' 'CP0125' 'CP0134' 'CP0137'
 'CP0151' 'CP0156' 'CP0157' 'CP0158' 'CP0159' 'CP0160' 'CP0163' 'CP0170'
 'CP0256' 'DAB001' 'DAB002' 'DAB003' 'DAB004' 'DAB005' 'DAB006' 'DAB007'
 'DAB008' 'DAB009' 'DAB010' 'DAB011' 'DAB012' 'DAB013' 'ENT0115' 'ENT0116'
 'ENT120' 'ENT121' 'ICE002' 'ICE004' 'PZ0001' 'PZ0002' 'PZ0003' 'PZ0004'
 'PZ0005' 'PZ0006' 'PZ0007' 'PZ0008' 'PZ0009' 'PZ0010' 'PZ0011' 'PZ0012'
 'PZ0013' 'PZ0014' 'PZ0015' 'PZ0016' 'PZ0017' 'PZ0018' 'PZ0019' 'PZ0020'
 'PZ0021' 'SAL100' 'SAL102' 'SAL103' 'SD0001' 'SUP121']
```

```
 'Raspberry, blueberry, strawberry, scoop of natural yoghurt ice cream, scoop of fruit yoghurt ice cream, raspberry sauce, choc
olate curls, wafer, whipped cream'
 'Romaine lettuce, parmesan, croutons & vinaigrette maison'
 'Rosemary, Olive Oil and Sea salt '
 'Sauteed broccoli, tomatoes, & cheese'
 'Sauteed chicken, broccoli & cheddar' 'Sauteed in a butter cream sauce'
 'Sauteed or grilled served with a side of broccoli, rice &, cream dill reduction'
 'Scallops & shrimp in a bechamel sauce with a side of rice'
 'Scrambled eggs, spinach & cheddar' 'Swiss and cheddar cheese'
 'Teriyaky Salmon & Shitake Mushrooms, with salad'
 'Tomato base, Cheddar cheese, bacon & eggs '
 'Tomato base, Folded pizza, mozzarella, mushrooms & ham '
 'Tomato base, Mozzarella & pepperoni '
 'Tomato base, Mozzarella, Ricotta, Gorgonzola & Parmesan shavings '
 'Tomato base, Mozzarella, black olives capers & anchovies '
 'Tomato base, Mozzarella, cheese & oregano '
 'Tomato base, Mozzarella, chorizo & Gorgonzola '
 'Tomato base, Mozzarella, chorizo & green pepper '
 'Tomato base, Mozzarella, ham & mushrooms '
 'Tomato base, Mozzarella, ham & pineapple '
 'Tomato base, Mozzarella, mushrooms, artichokes, black olives & ham '
 'Tomato base, Mozzarella, mushrooms, ham, green pepper & red onion '
 'Tomato base, Mozzarella, mushrooms, pepper, aubergine, olives & garlic '
 'Tomato base, Mozzarella, spinach & egg '
 'Tomato base, Mozzarella, tuna & onion '
 'Tomato base, Ricotta cheese, spinach & black pepper '
 'Tomato base, red onions, anchovies, capers, olives & chilli '
 'Traditional French onion soup with bread & melted emmenthal'
 'Tuna, black olives, cheddar cheese, spinach and our special tomato sauce!'
 'Tuna, potatoes, green beans, onions, capers, tomatoes, egg and anchovies on a bed of romaine lettuce with the vinaigrette mai
son'
 'Vodka, Tomato Juice and Tabasco - what elsee' 'Your choice!' 'nan']

Unique values in column 'Price': 28
[ 3.    3.25  3.5   3.75  4.    4.5   5.    5.5   5.75  6.5   6.75  7.
  7.5   8.    8.5   9.    9.5  10.   10.5  11.   11.5  11.75 12.5  13.
 13.5  14.   14.5  16.  ]

Unique values in column 'Product_Group': 9
['A   ' 'B   ' 'C' 'D   ' 'F   ' 'G   ' 'K   ' 'P   ' 'S   ']

Unique values in column 'Group_name': 9
['Breakfast' 'Crepes and Galettes' 'Desserts' 'Drinks & Beverages'
 'Fish & Seafood' 'Pizza' 'Salads' 'Side Dishes' 'Starters/Appetisers']

Unique values in column 'CardType': 4
['Avis' 'Debit Card' 'Maestercard' 'Visa']
```

Figure 6: Checking rest of data.

After cleaning the data, it is verify if it has been cleaned.

Strip the white space of each column:

```
# goes through each column and strips whitespaces
for col_name in df.columns:
    if df[col_name].dtypes == object:
        df[col_name] = df[col_name].str.strip()
    else:
        pass
```

Look at sample data to see if it was successful

```
np.unique(df['Product_Group'])
```

```
array(['A', 'B', 'C', 'D', 'F', 'G', 'K', 'P', 'S'], dtype=object)
```

Figure 7: Cleaning white space and verifying it.

Looking at the variable $TicketNo$, it is a variable which indicates a sale from a customer and there seems to be duplicates, but looking at the data it is just one customer buying more than one item as shown in figure 8.

```
df[['TicketNo']].duplicated().sum()
```

```
23997
```

```
df[df['TicketNo'] == 92217]
```

	SaleDate	TicketNo	Outlet	Total	OrderQty	Stock_Code	Name	Description	Price	Product_Group	Group_name	CardType
3	2018-08-21	92217	Birmingham	5.0	1	BK0101	Eggs Florentine	NaN	5.0	B	Breakfast	Avis
4	2018-08-21	92217	Birmingham	7.5	1	CP0125	The Sicilian Crepe	Tuna, black olives, cheddar cheese, spinach an...	7.5	C	Crepes and Galettes	Avis

One ticket number shows the different items bought for that particular ticket/person, but not a duplicate

Figure 8: TicketNo inspection.

The data is cleaned in a way such that it is reproducible if any new data is to be inserted into this data set. Currently the data set has 40,705 which is not a lot of data and hence python would be able to handle the initial stage of cleaning. For future proof, if new rows are being inserted, let the quantity be such that python can handle it i.e. if data is too large, load it in batches or insert data more frequently OR simply insert records into the data correctly. This initial stage of cleaning produces a foundation for the Galleria operations, whereby this data set can be imported to SQL by connecting to the database as shown in figure 9.

```
import pyodbc
from sqlalchemy import create_engine
import urllib

SERVER_NAME = 'DESKTOP-GR6T3L7'
DATABASE_NAME = 'Galleria'

# connection string
conn = pyodbc.connect('DRIVER={ODBC Driver 17 for SQL Server}; \
                       SERVER=' + SERVER_NAME + '; \
                       DATABASE=' + DATABASE_NAME + '; \
                       Trusted_Connection=yes')

# cursor to extract data
cursor = conn.cursor()

# Use this to connect for saving data
quoted = urllib.parse.quote_plus('DRIVER={ODBC Driver 17 for SQL Server}; \
                       SERVER='+SERVER_NAME+'; \
                       DATABASE='+DATABASE_NAME+'; \
                       Trusted_Connection=yes')

# save the dataset into SQL database
engine = create_engine('mssql+pyodbc:///?odbc_connect={}'.format(quoted))
df.to_sql('initial_dataset', schema='dbo', con=engine, if_exists='replace')

# close port
cursor.close()
conn.close(
)
print('Dataset appended.')
```

```
Dataset appended.
```

Figure 9: Importing cleaned data into SQL database.

3.2 Exploratory Data Analysis

This section consists of looking at each variable and how they relate to each other and trying to identify relationships and trends as well as looking at the distribution of the data.

3.2.1 Numerical Data

One of the most common ways to look at how data is distributed is to use central tendency.

```
data.describe()
```

	TicketNo	Total	OrderQty	Price
count	40705.000000	40705.000000	40705.000000	40705.000000
mean	92232.059526	18.237716	2.462769	7.590910
std	9052.340917	18.114654	2.391667	2.608612
min	82191.000000	3.000000	1.000000	3.000000
25%	85270.000000	7.500000	1.000000	5.000000
50%	88650.000000	10.000000	1.000000	7.500000
75%	98323.000000	22.500000	3.000000	9.500000
max	114781.000000	216.000000	32.000000	16.000000

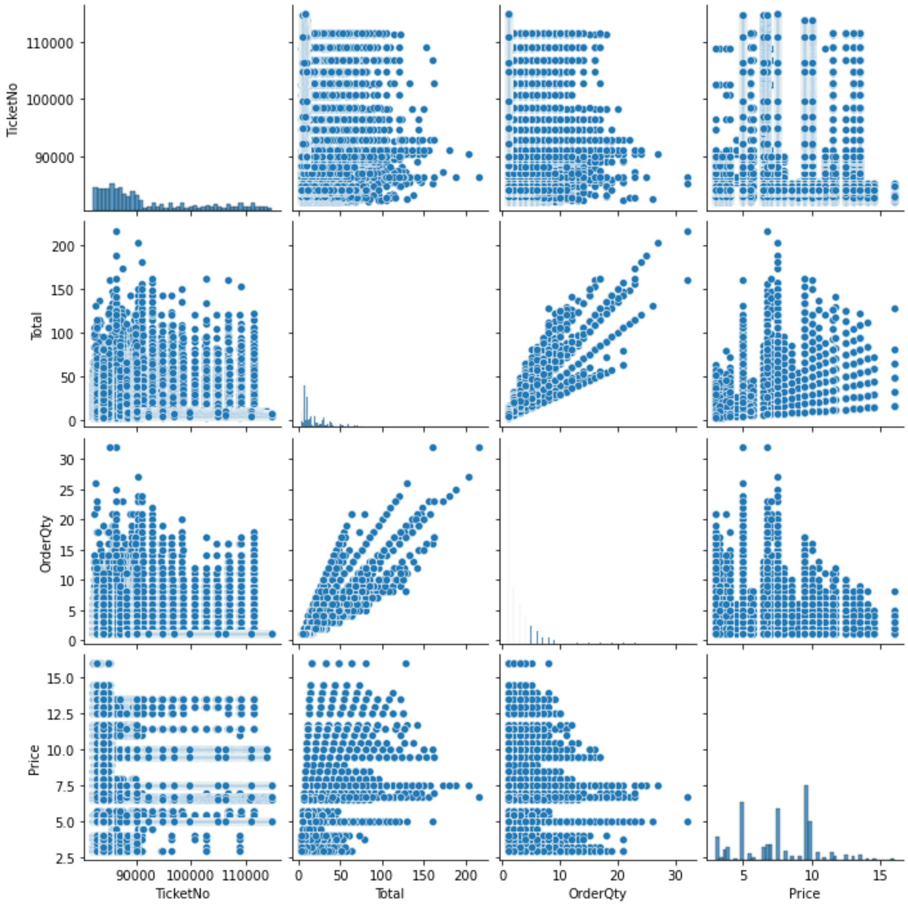
Notes:

- The most important variable for a business is to maximise profits and by looking at the variable *total*, we can understand how it varies.
- The order quantity ranges from 1 to 32 for one ticket but the mean is around 2.5 orders per item.
- The mean spent from one ticket is around £18, the minimum is £3 and the maximum is £216, where the £216 could be an anomaly.
- The mean price is around £7.50, the minimum is £2.60 and the maximum is £16.00 which which means is roughly the in between the min and max.

Figure 10: Data distribution.

A better way of understanding the data is by looking at how each variable are correlated with each other on a graph as shown in figure 12.

```
sns.pairplot(data)
plt.show()
```



Notes:

- Order quantity is lower for higher prices and generally a higher order quantity is positively correlated with the total amount spent.

Figure 11: Pairplot showing relationship of all variables.

As revenue is one of the most important factors when considering the success of a business, it would be useful to see how revenue is affected by other variables and this is demonstrated in figure 12

```python
total_correlation = data.corr()['Total'].sort_values(ascending=False)

with sns.axes_style("white"):
    plt.figure(figsize=(7,4))
    total_correlation.plot.bar()
    plt.ylabel('Correlation')
    plt.title('Total Spent Correlation vs Variables')
    plt.show()
```

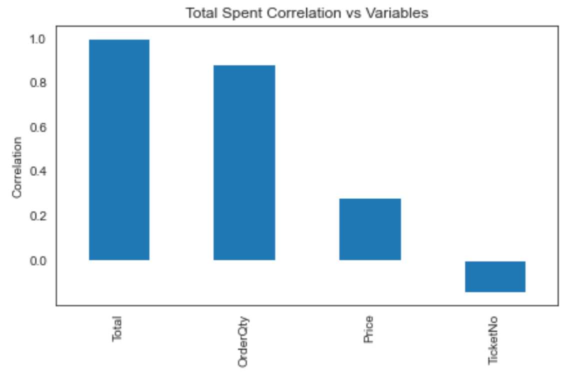


Figure 12: Correlation of revenue and numerical data.

3.2.2 Categorical Data

It is equally important to look at the qualitative variables that has an affect on the data. One measure of determining the performance of Galleria is to see how their revenue varies over time. Figure 16 shows how revenue changes over the course of a year.

```python
from datetime import datetime

data['SaleDate'] = pd.to_datetime(data['SaleDate'])
```

Looking at how sales vary with month and year

```python
data['SaleMonth'] = data['SaleDate'].dt.month
```

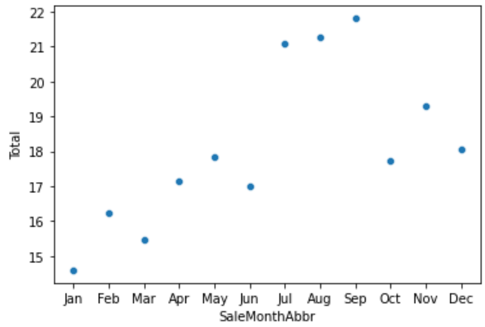
```python
import calendar

data['SaleMonthAbbr'] = data['SaleMonth'].apply(lambda x: calendar.month_abbr[x])
```

```python
by_month = data.groupby(['SaleMonth', 'SaleMonthAbbr']).agg({'Total': 'mean', 'OrderQty':'mean'})
by_month
```

		Total	OrderQty
SaleMonth	SaleMonthAbbr		
1	Jan	14.596248	2.000816
2	Feb	16.225779	2.203875
3	Mar	15.485018	2.091867
4	Apr	17.136807	2.353277
5	May	17.841837	2.401743
6	Jun	16.987483	2.284503
7	Jul	21.092904	2.870885
8	Aug	21.282662	2.864440
9	Sep	21.804009	2.890772
10	Oct	17.717975	2.444650
11	Nov	19.287005	2.593820
12	Dec	18.069911	2.409788

```
sns.scatterplot(x='SaleMonthAbbr', y='Total', data=by_month)
```

```
<AxesSubplot:xlabel='SaleMonthAbbr', ylabel='Total'>
```



- Sales usually increase throughout the year but more significantly during the summer.

Figure 13: Revenue vs Month of Year

Likewise, the yearly performance can also be observed to see how Galleria have been performing over the course of their time in the UK.

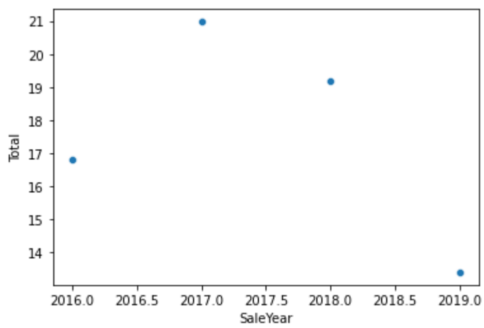
```
data['SaleYear'] = data['SaleDate'].dt.year
```

```
by_year = data.groupby('SaleYear').agg({'Total': 'mean', 'OrderQty':'mean'})
by_year
```

	Total	OrderQty
SaleYear		
2016	16.814440	2.300020
2017	20.976292	2.860546
2018	19.200004	2.581745
2019	13.409238	1.770092

```
sns.scatterplot(x='SaleYear', y='Total', data=by_year)
```

```
<AxesSubplot:xlabel='SaleYear', ylabel='Total'>
```



- Sales started to decrease after 2017.

Figure 14: Revenue vs Year

Additionally, looking at the variable *Outlet* and *GroupCategory*, it can be seen how they vary with revenue.

```
by_location = data.groupby('Outlet').agg({'Total': 'mean', 'OrderQty':'mean'}).reset_index().sort_values('Total', ascending=
by_location
```

	Outlet	Total	OrderQty
4	London	23.179106	3.148175
9	Worthing	22.743359	3.156114
1	Cardiff	21.952769	3.025335
2	Edinburgh	21.201490	2.896703
5	Middlesborough	18.801459	2.559357
0	Birmingham	18.801087	2.420227
7	Poole	16.694711	2.253094
3	Ipswich	14.910027	1.968265
8	Weymouth	11.464674	1.601449
6	Peterborough	8.971286	1.161548

```
plt.figure(figsize=(12,8))
sns.barplot(x='Outlet', y='Total', data=by_location)
plt.xticks(rotation=45)
plt.show()
```

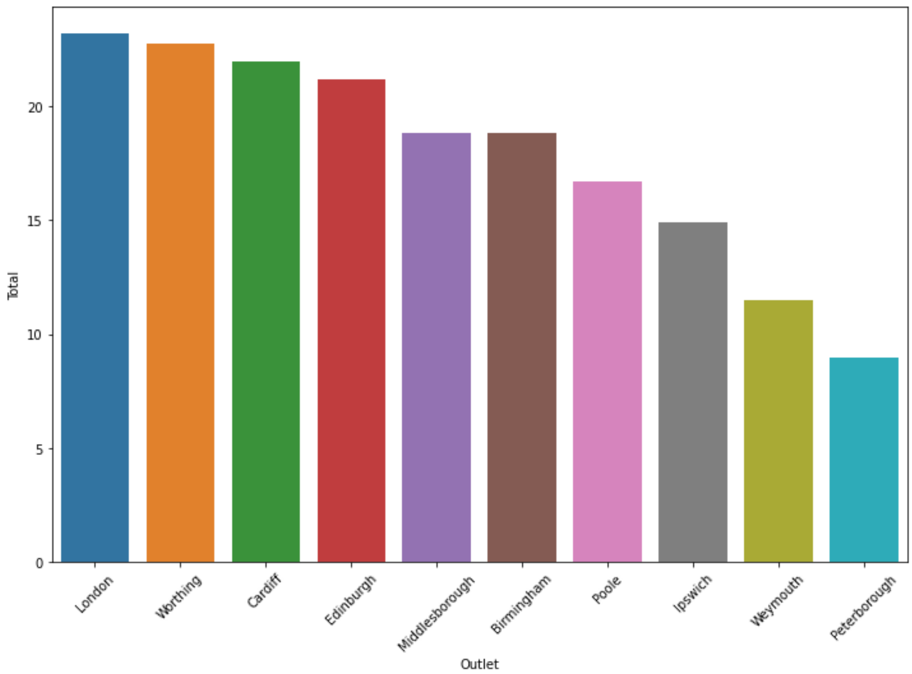


Figure 15: Revenue vs Location

```
by_product = data.groupby('Group_name').agg({'Total': 'mean', 'OrderQty':'mean'}).reset_index().sort_values('Total', ascendi
by_product
```

	Group_name	Total	OrderQty
8	Starters/Appetisers	31.323877	2.591017
4	Fish & Seafood	26.211165	1.849515
1	Crepes and Galettes	22.279510	2.811535
5	Pizza	21.965373	2.306699
6	Salads	21.357386	2.439773
2	Desserts	15.740754	1.856191
0	Breakfast	12.261673	2.326537
7	Side Dishes	11.924409	2.981102
3	Drinks & Beverages	10.997214	2.770734

```
plt.figure(figsize=(12,8))
sns.barplot(x='Group_name', y='Total', data=by_product)
plt.xticks(rotation=45)
plt.show()
```

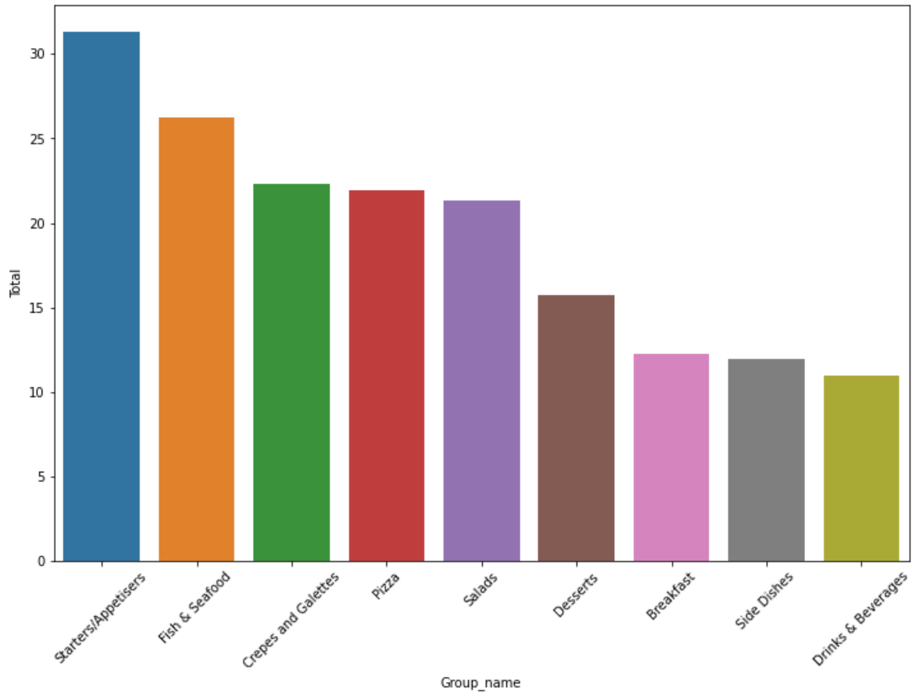


Figure 16: Revenue vs Group Product

4 OLAP Schema Design

4.1 Looking at the Data

An Online Analytical Processing (OLAP) schema is schema in which it is designed for analytical processing. As the project is to build a warehouse that is capable of analysing the sales data for Galleria, an OLAP schema is suitable for this. Additionally, there are two common types of OLAP schemas designs; the star and snowflake models. A star schema is one whereby there is one fact table and multiple dimension tables, where each dimension table only is linked to the fact table. In a snowflake schema, there can be sub dimension tables joined to dimension tables so a dimension table can have many links.

For the data warehouse that is to be created, the most suitable schema is the star schema as it is an easy to understand schema which makes it easy and quick to query. Also, it is the recommended schema to use when working with PowerBI.

When deciding how to split the initial data set to create a fact to dimension table relationship, a look at how the variables are related to each other can help doing so.

```
df.head(2)
```

	SaleDate	TicketNo	Outlet	Total	OrderQty	Stock_Code	Name	Description	Price	Product_Group	Group_name	CardType
0	2018-08-21	92208	Birmingham	5.0	1	BK0101	Eggs Florentine	NaN	5.0	B	Breakfast	Visa
1	2018-08-21	92209	Birmingham	7.5	1	CP0125	The Sicilian Crepe	Tuna, black olives, cheddar cheese, spinach an...	7.5	C	Crepes and Galettes	Avis

```
np.unique(df[['Stock_Code']].values)
```

```
array(['APP001', 'APP002', 'BK0101', 'BK0102', 'BK0103', 'BK0110',
       'BK0160', 'BK0161', 'BK0162', 'CP0101', 'CP0102', 'CP0105',
       'CP0107', 'CP0108', 'CP0110', 'CP0111', 'CP0112', 'CP0113',
       'CP0117', 'CP0118', 'CP0119', 'CP0125', 'CP0134', 'CP0137',
       'CP0151', 'CP0156', 'CP0157', 'CP0158', 'CP0159', 'CP0160',
       'CP0163', 'CP0170', 'CP0256', 'DAB001', 'DAB002', 'DAB003',
       'DAB004', 'DAB005', 'DAB006', 'DAB007', 'DAB008', 'DAB009',
       'DAB010', 'DAB011', 'DAB012', 'DAB013', 'ENT0115', 'ENT0116',
       'ENT120', 'ENT121', 'ICE002', 'ICE004', 'PZ0001', 'PZ0002',
       'PZ0003', 'PZ0004', 'PZ0005', 'PZ0006', 'PZ0007', 'PZ0008',
       'PZ0009', 'PZ0010', 'PZ0011', 'PZ0012', 'PZ0013', 'PZ0014',
       'PZ0015', 'PZ0016', 'PZ0017', 'PZ0018', 'PZ0019', 'PZ0020',
       'PZ0021', 'SAL100', 'SAL102', 'SAL103', 'SD0001', 'SUP121'],
      dtype=object)
```

```
df[df['Stock_Code'] == 'PZ0021'].head(3)
```

	SaleDate	TicketNo	Outlet	Total	OrderQty	Stock_Code	Name	Description	Price	Product_Group	Group_name	CardType
87	2018-08-25	92399	Ipswich	10.0	1	PZ0021	THE DON	(Folded pizza) Tomato base, mushrooms, red oni...	10.0	P	Pizza	Maestercard
160	2018-08-28	92569	Poole	10.0	1	PZ0021	THE DON	(Folded pizza) Tomato base, mushrooms, red oni...	10.0	P	Pizza	Debit Card
216	2018-08-30	92667	Peterborough	10.0	1	PZ0021	THE DON	(Folded pizza) Tomato base, mushrooms, red oni...	10.0	P	Pizza	Debit Card

- Stock Code can be used to identify which product is being sold i.e. the name of the product so it can be used in a new table. Alongside this, the description and price can be linked.

- The product group can be used to identify which group the product lies in and can be linked with the group name.

- The outlet can have its own table as it is just a city. Likewise, the card type can be split in the same manner.

Figure 17: Looking at the variables and deciding how to split the data.

4.2 Design of Star Schema

The first step in creating a star schema from the initial data set is to create a primary key for each dimension table that will all be inside the fact table as foreign keys to join the dimension tables to the fact table. The dimension table is split by grouping similar data together and this is shown in figure 18.

Fact Table:

- [SaleID, SaleDate, TicketNo, Stock_Code, OutletID, CardID, GroupID, OrderQty, Total] --> SalesID is primary key

Dimension Tables:

- - [GroupID, Product_Group, Group_name]
  - [StockCode, Name, Description, Price]
  - [OutletID, Outlet]
  - [CardID, CardType]

Figure 18: Fact and dimension splitting of variables

A data warehouse needs to be stored in a database and hence, the reason why the data is imported from Jupyter to Microsoft SQL Server. A database is capable of storing a large amount of data, and suitable in creating an OLAP star schema. Therefore, by writing the following queries, a star schema can be created.

```sql
/***

CREATE A STAR SCHEMA BASED ON THE PLAN CREATED:

Fact Table:
- [SaleID, SaleDate, TicketNo, Stock_Code, OutletID, CardID, ProductGroup, OrderQty, 
  Total] --> SalesID is primary key

Dimension Tables:
-     - [Product_Group, Group_name]
    - [StockCode, Name, Description, Price]
    - [OutletID, Outlet]
    - [CardID, CardType]

***/

CREATE DATABASE Galleria

USE Galleria
GO

---------- DROP TABLE TO ENSURE CREATION IF EXISTS ----------
BEGIN
    DROP TABLE dbo.SalesFact
    DROP TABLE dbo.GroupDimension
    DROP TABLE dbo.NameDimension
    DROP TABLE dbo.OutletDimension
    DROP TABLE dbo.CardDimension
    PRINT 'Tables Dropped'
END

---------- CREATION AND INSERTION OF DIMENSION TABLES ----------

SELECT * FROM dbo.initial_dataset

-- 1A CREATION OF GROUP TABLE
CREATE TABLE GroupDimension
(
    ProductGroup VARCHAR(5) NOT NULL PRIMARY KEY,
    GroupName VARCHAR(50),
    CreateTimestamp DATETIME,
    UpdateTimestamp DATETIME
)

-- 1B INSERTION OF GROUP TABLE
INSERT INTO GroupDimension
SELECT DISTINCT ProductGroup,
GroupName,
CURRENT_TIMESTAMP AS CreateTimestamp,
CURRENT_TIMESTAMP AS UpdateTimestamp
FROM Galleria.dbo.initial_dataset
```

```sql
-- 2A CREATION OF NAME TABLE
CREATE TABLE NameDimension
(
    StockCode VARCHAR(50) NOT NULL PRIMARY KEY, -- Already a suitable key
    Name VARCHAR(100),
    Description VARCHAR(255),
    Price INT NOT NULL,
    CreateTimestamp DATETIME,
    UpdateTimestamp DATETIME
)

-- 2B INSERTION OF NAME TABLE
INSERT INTO NameDimension
SELECT DISTINCT StockCode,
Name,
Description,
Price,
CURRENT_TIMESTAMP AS CreateTimestamp,
CURRENT_TIMESTAMP AS UpdateTimestamp
FROM Galleria.dbo.initial_dataset


-- 3A CREATION OF OUTLET TABLE
CREATE TABLE OutletDimension
(
    OutletID INT IDENTITY(1,1) NOT NULL PRIMARY KEY,
    Outlet VARCHAR(100),
    CreateTimestamp DATETIME,
    UpdateTimestamp DATETIME
)

-- 3B INSERTION INTO OUTLET TABLE
INSERT INTO OutletDimension
SELECT DISTINCT Outlet,
CURRENT_TIMESTAMP AS CreateTimestamp,
CURRENT_TIMESTAMP AS UpdateTimestamp
FROM Galleria.dbo.initial_dataset


-- 4A CREATION OF CARD TABLE
CREATE TABLE CardDimension
(
    CardID INT IDENTITY(1,1) NOT NULL PRIMARY KEY,
    CardType VARCHAR(20),
    CreateTimestamp DATETIME,
    UpdateTimestamp DATETIME
)

-- 4B INSERTION INTO CARD TABLE
INSERT INTO CardDimension
SELECT DISTINCT CardType,
CURRENT_TIMESTAMP AS CreateTimestamp,
```

```
CURRENT_TIMESTAMP AS UpdateTimestamp
FROM Galleria.dbo.initial_dataset


---------- CREATION AND INSERTION OF FACT TABLE ----------

-- CREATION OF FACT TABLE
CREATE TABLE SalesFact
(
    SaleID INT NOT NULL IDENTITY(1,1) PRIMARY KEY,
    SaleDate DATE NOT NULL,
    TicketNo INT NOT NULL,
    StockCode VARCHAR(50) NOT NULL FOREIGN KEY REFERENCES NameDimension(StockCode),
    ProductGroup VARCHAR(5) NOT NULL FOREIGN KEY REFERENCES GroupDimension
      (ProductGroup),
    OutletID INT NOT NULL FOREIGN KEY REFERENCES OutletDimension(OutletID),
    CardID INT NOT NULL FOREIGN KEY REFERENCES CardDimension(CardID),
    OrderQty INT NOT NULL,
    Total INT NOT NULL,
    CreateTimestamp DATETIME,
    UpdateTimestamp DATETIME

)

-- INSERTION OF FACT TABLE
INSERT INTO SalesFact
SELECT
    a.SaleDate,
    a.TicketNo,
    a.StockCode,
    a.ProductGroup,
    c.OutletID,
    d.CardID,
    a.OrderQty,
    a.Total,
    CURRENT_TIMESTAMP AS CreateTimestamp,
    CURRENT_TIMESTAMP AS UpdateTimestamp
FROM initial_dataset a
INNER JOIN GroupDimension b ON a.ProductGroup = b.ProductGroup
    INNER JOIN OutletDimension c ON a.Outlet = c.Outlet
        INNER JOIN CardDimension d ON a.CardType = d.CardType


---------- VERIFICATION ----------

-- View Tables created

SELECT *
FROM GroupDimension

SELECT *
FROM OutletDimension
```

```sql
SELECT *
FROM NameDimension

SELECT *
FROM CardDimension

SELECT *
FROM SalesFact

-- VERIFICATION OF SUCCRSSFUL SCHEMA
SELECT salesfact.saleid,
       salesfact.saledate,
       salesfact.ticketno,
       salesfact.stockcode,
       salesfact.orderqty,
       salesfact.total,
       namedimension.NAME,
       namedimension.description,
       namedimension.price,
       carddimension.cardtype,
       groupdimension.groupname,
       outletdimension.outlet
FROM   carddimension
       INNER JOIN salesfact
               ON carddimension.cardid = salesfact.cardid
       INNER JOIN groupdimension
               ON salesfact.productgroup = groupdimension.productgroup
       INNER JOIN namedimension
               ON salesfact.stockcode = namedimension.stockcode
       INNER JOIN outletdimension
               ON salesfact.outletid = outletdimension.outletid
WHERE TicketNo = 92785

SELECT *
FROM initial_dataset
WHERE TicketNo = 92785

-- BY COUNt
SELECT COUNT(*)
FROM SalesFact

SELECT COUNT(*)
FROM NameDimension

SELECT COUNT(*)
FROM GroupDimension

SELECT COUNT(*)
FROM CardDimension

SELECT COUNT(*)
```

```sql
FROM OutletDimension

SELECT COUNT(*)
FROM initial_dataset

-- END
```

A database diagram can better illustrate the creation of the OLAP schema which is seen in figure 19

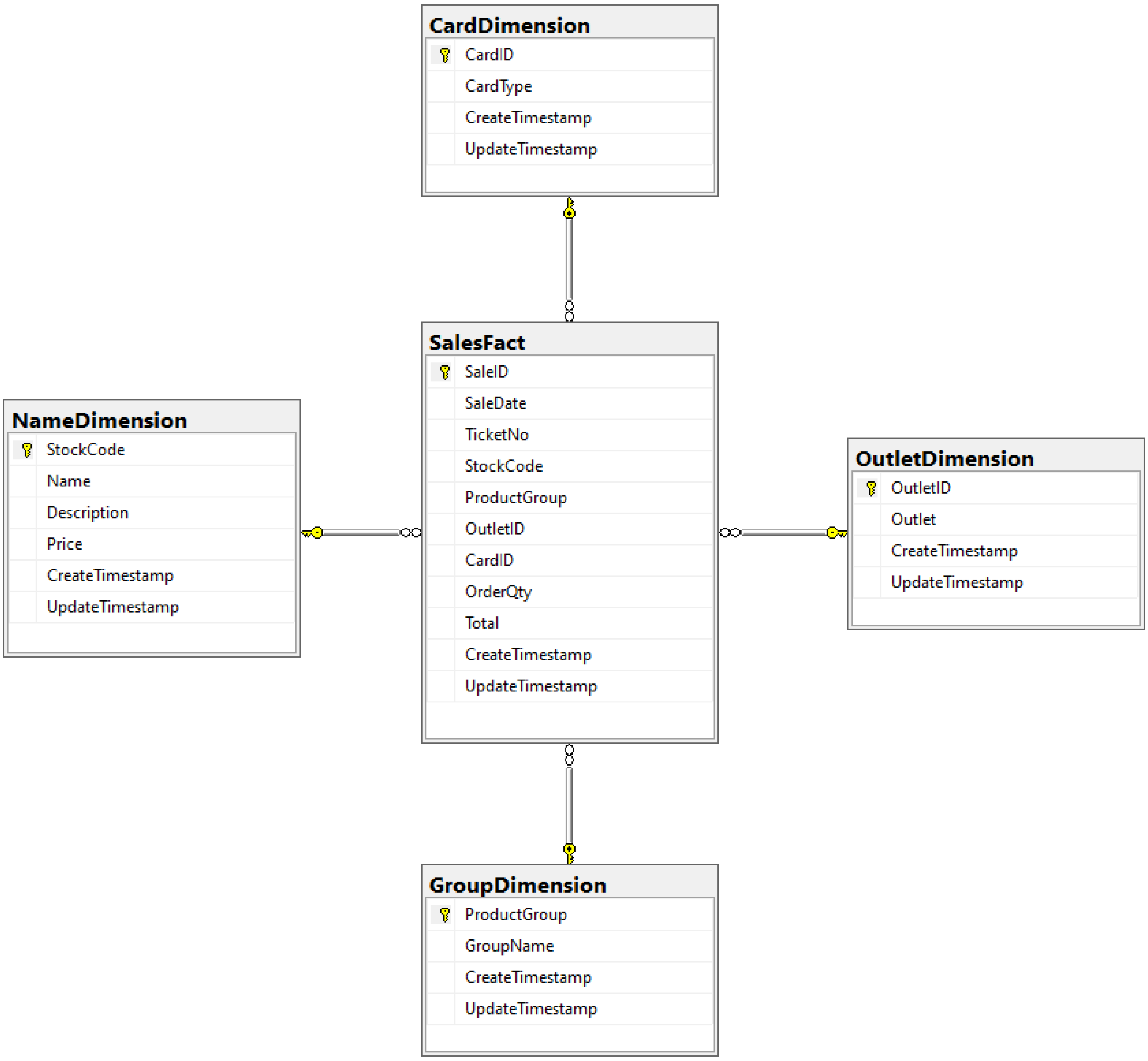


Figure 19: Database diagram.

Note: Timestamps were also added to each table. These are used in data warehouses to keep track of when records were loaded and updated in the data warehouses (as opposed to when they were created or updated in the operational source systems).

5 Source to Target Mapping

A source to target mapping is a blueprint of the ETL solution as it shows how the initial source data set is split into mulitple data sets that are related together through primary and foreign keys. It can be seen in figure 20.

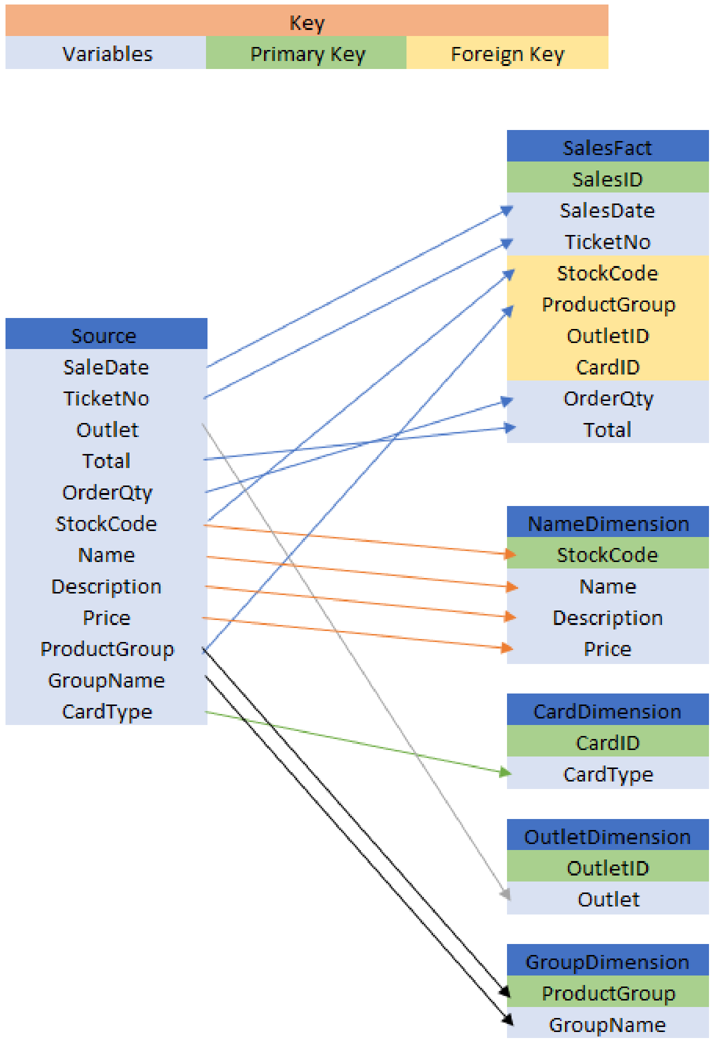


Figure 20: Source to target mapping.

Note that the CardID and OutletID are new columns created to act as a primary key for the dimension tables. Hence, they were not mapped from the source but rather, were created to complete the star schema.

6 Test Approach

To make sure the data set is split correctly i.e. the relation between the fact table and dimension tables are correct and the keys match the correct row, some testing of the data must be carried out to demonstrate this. By taking a sample of the data from the fact table joined with the dimension tables and comparing it with the initial data set, the merge can be identified as either successful or unsuccessful. This is shown in figure 21.

```sql
-- VERIFICATION OF SUCCRSSFUL SCHEMA
SELECT salesfact.saleid,
       salesfact.saledate,
       salesfact.ticketno,
       salesfact.stockcode,
       salesfact.orderqty,
       salesfact.total,
       namedimension.NAME,
       namedimension.description,
       namedimension.price,
       carddimension.cardtype,
       groupdimension.groupname,
       outletdimension.outlet
FROM   carddimension
       INNER JOIN salesfact
               ON carddimension.cardid = salesfact.cardid
       INNER JOIN groupdimension
               ON salesfact.productgroup = groupdimension.productgroup
       INNER JOIN namedimension
               ON salesfact.stockcode = namedimension.stockcode
       INNER JOIN outletdimension
               ON salesfact.outletid = outletdimension.outletid
WHERE TicketNo = 92785

SELECT *
FROM initial_dataset
WHERE TicketNo = 92785
```

	saleid	saledate	ticketno	stockcode	orderqty	total	NAME	description	price	cardtype	groupname	outlet
1	319	2018-09-01	92785	PZ0019	1	10	MOO-MOO	Tomato base, Mozzarella, chorizo & Gorgonzola	10	Avis	Pizza	Poole

	index	SaleDate	TicketNo	Outlet	Total	OrderQty	StockCode	Name	Description	Price	ProductGroup	GroupName	CardType
1	318	2018-09-01	92785	Poole	10	1	PZ0019	MOO-MOO	Tomato base, Mozzarella, chorizo & Gorgonzola	10	P	Pizza	Avis

Figure 21: Verification of OLAP star schema.

Additionally, a count of the rows for each table must be the same for each table to show a successful merge and this can be seen in the TSQL code shown in section 4.2. Also, from section 4.2 it can be seen that by running a query to view all the new tables, the success of the creation can be determined by looking at the number of values in each table as seen in figure 22

Results | Messages

	ProductGroup	GroupName	CreateTimestamp	UpdateTimestamp
2	B	Breakfast	2022-08-11 15:08:10.543	2022-08-11 15:08:10.543
3	C	Crepes and Galettes	2022-08-11 15:08:10.543	2022-08-11 15:08:10.543
4	D	Desserts	2022-08-11 15:08:10.543	2022-08-11 15:08:10.543
5	F	Fish & Seafood	2022-08-11 15:08:10.543	2022-08-11 15:08:10.543
6	G	Drinks & Beverages	2022-08-11 15:08:10.543	2022-08-11 15:08:10.543
7	K	Side Dishes	2022-08-11 15:08:10.543	2022-08-11 15:08:10.543
8	P	Pizza	2022-08-11 15:08:10.543	2022-08-11 15:08:10.543
9	S	Salads	2022-08-11 15:08:10.543	2022-08-11 15:08:10.543

	OutletID	Outlet	CreateTimestamp	UpdateTimestamp
1	1	Worthing	2022-08-11 15:08:19.897	2022-08-11 15:08:19.897
2	2	Poole	2022-08-11 15:08:19.897	2022-08-11 15:08:19.897
3	3	Peterborough	2022-08-11 15:08:19.897	2022-08-11 15:08:19.897
4	4	London	2022-08-11 15:08:19.897	2022-08-11 15:08:19.897
5	5	Middlesborough	2022-08-11 15:08:19.897	2022-08-11 15:08:19.897
6	6	Birmingham	2022-08-11 15:08:19.897	2022-08-11 15:08:19.897
7	7	Weymouth	2022-08-11 15:08:19.897	2022-08-11 15:08:19.897
8	8	Edinburgh	2022-08-11 15:08:19.897	2022-08-11 15:08:19.897

	StockCode	Name	Description	Price	CreateTimestamp	UpdateTimestamp
1	APP001	Assiette de pate	3 pates with cornichons & moutarde	12	2022-08-11 15:08:15.337	2022-08-11 15:08:15.337
2	APP002	Assiette de fromages	Monthy selection of 3 cheeses	11	2022-08-11 15:08:15.337	2022-08-11 15:08:15.337
3	BK0101	Eggs Florentine	NULL	5	2022-08-11 15:08:15.337	2022-08-11 15:08:15.337
4	BK0102	Eggs Benedict	NULL	5	2022-08-11 15:08:15.337	2022-08-11 15:08:15.337
5	BK0103	Eggs Royale	NULL	5	2022-08-11 15:08:15.337	2022-08-11 15:08:15.337
6	BK0110	Belgian waffle	NULL	5	2022-08-11 15:08:15.337	2022-08-11 15:08:15.337
7	BK0160	chicken 'n cheese ...	Sauteed chicken, broccoli & ched...	8	2022-08-11 15:08:15.337	2022-08-11 15:08:15.337
8	BK0161	Scrambled eggs	Scrambled eggs, spinach & ched...	8	2022-08-11 15:08:15.337	2022-08-11 15:08:15.337

	CardID	CardType	CreateTimestamp	UpdateTimestamp
1	1	Debit Card	2022-08-11 15:08:24.043	2022-08-11 15:08:24.043
2	2	Maestercard	2022-08-11 15:08:24.043	2022-08-11 15:08:24.043
3	3	Avis	2022-08-11 15:08:24.043	2022-08-11 15:08:24.043
4	4	Visa	2022-08-11 15:08:24.043	2022-08-11 15:08:24.043

	SaleID	SaleDate	TicketNo	StockCode	ProductGroup	OutletID	CardID	OrderQty	Total	CreateTimestamp	UpdateTimestamp
4...	40698	2019-03-09	105380	CP0170	C	5	1	1	7	2022-08-11 15:08:29.427	2022-08-11 15:08:29.427
4...	40699	2019-03-09	105384	BK0110	B	5	3	1	5	2022-08-11 15:08:29.427	2022-08-11 15:08:29.427
4...	40700	2019-03-09	105386	BK0101	B	7	1	1	5	2022-08-11 15:08:29.427	2022-08-11 15:08:29.427
4...	40701	2019-03-09	105387	BK0101	B	9	2	1	5	2022-08-11 15:08:29.427	2022-08-11 15:08:29.427
4...	40702	2019-03-10	105396	CP0170	C	8	3	1	7	2022-08-11 15:08:29.427	2022-08-11 15:08:29.427
4...	40703	2019-03-10	105397	CP0134	C	8	2	1	7	2022-08-11 15:08:29.427	2022-08-11 15:08:29.427
4...	40704	2019-03-10	105401	BK0110	B	3	3	1	5	2022-08-11 15:08:29.427	2022-08-11 15:08:29.427
4...	40705	2019-03-10	105401	CP0137	C	3	3	1	6	2022-08-11 15:08:29.427	2022-08-11 15:08:29.427

Figure 22: Verification of OLAP star schema.

7 Visualisations

Using PowerBI, a dashboard can be created to summarise all the business questions asked in an easy to understand manner. Galleria required an analysis in which they can determine the most popular items on their menu and the ones which produce the most revenue. This can be seen under Menu Items on the dashboard on the following page.

Additionally, Galleria want to look at best selling items and also the worst selling items, so they can sell more of their best selling items and cut down some of the worst selling items on the menu to consolidate their menu and only provide items that the customers prefer. Under location on the dashboard, the best selling items can be seen for each location and by filtering the data using the filters, the worst items can be determined by order quantity and revenue, which Galleria can decide to cut out.

Furthermore, Galleria want to determine which product group is generating the most revenue, and this can be seen under product categories on the dashboard.

Moreover, a league table of each outlet showing their revenue performances for each month is shown under the section league table, where it can also be filtered by the week number of the year.

Additional analysis which could benefit Galleria is looking out how the sales vary through the year and see how they can capitalise on time period where it is most busy i.e. more customers and hence more revenue.

Galleria Analytics Dashboard

Location: All

Product Category: All

Price: All

Card Type: All

1123
Count of Sale Dates

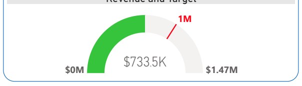
Outlet

16.71K
Number of Customers

Sale Date

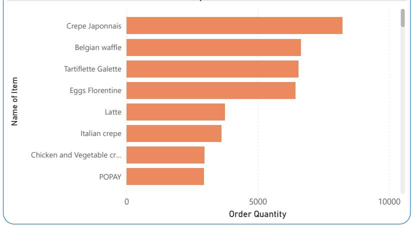
01/07/2016 31/07/2019

Revenue and Target

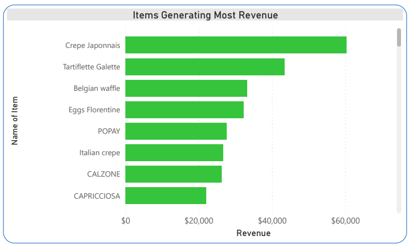


Menu Items

Most Popular Items

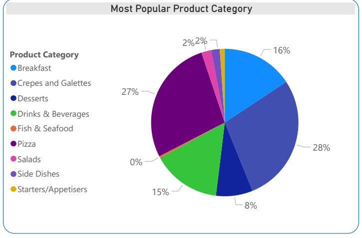


Items Generating Most Revenue

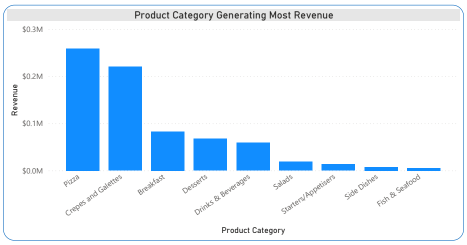


Product Categories

Most Popular Product Category

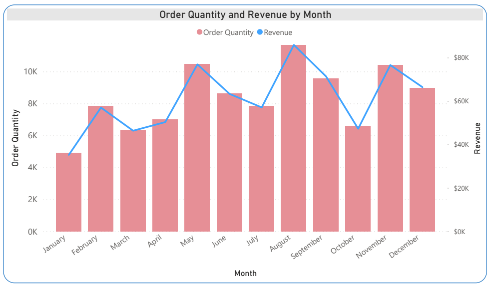


Product Category Generating Most Revenue

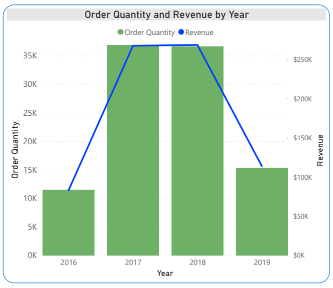


Time of Year

Order Quantity and Revenue by Month



Order Quantity and Revenue by Year



League Tables

Monthly Sales of Each Location

Month	Birmingham	Cardiff	Edinburgh	Ipswich	London	Middlesborough	Peterborough	Poole	Weymouth	Worthing	Total
January	$4,880	$3,257	$6,557	$2,228	$2,361	$6,915	$2,752	$1,561	$1,442	$3,269	**$35,222**
February	$9,628	$4,849	$9,308	$2,730	$4,537	$12,215	$2,828	$4,256	$1,877	$4,766	**$56,994**
March	$4,786	$2,727	$8,858	$3,330	$4,397	$12,099	$3,141	$1,976	$1,516	$3,513	**$46,343**
April	$6,065	$5,057	$10,625	$2,861	$3,893	$9,919	$2,653	$2,498	$1,916	$4,825	**$50,312**
May	$12,920	$6,435	$13,427	$3,156	$6,509	$17,993	$2,726	$5,297	$2,114	$6,289	**$76,866**
June	$6,166	$4,215	$11,790	$4,573	$6,011	$17,460	$3,766	$2,722	$1,830	$4,755	**$63,288**
July	$7,051	$5,421	$12,430	$3,450	$5,127	$11,862	$2,121	$2,855	$1,792	$4,958	**$57,067**
August	$13,350	$7,670	$16,757	$4,315	$7,278	$17,649	$2,373	$6,503	$2,464	$7,470	**$85,829**
September	$6,765	$4,600	$13,097	$5,699	$5,870	$20,383	$3,930	$3,505	$1,506	$6,000	**$71,355**
October	$5,779	$4,421	$10,006	$2,878	$3,773	$9,766	$2,658	$1,917	$1,585	$4,483	**$47,266**
November	$13,731	$7,004	$13,334	$3,355	$6,834	$16,423	$2,207	$5,248	$1,973	$6,462	**$76,571**
December	$7,177	$4,510	$12,648	$4,869	$6,426	$17,295	$3,794	$2,879	$1,608	$5,183	**$66,389**
Total	**$98,298**	**$60,166**	**$138,837**	**$43,444**	**$63,016**	**$169,979**	**$34,949**	**$41,217**	**$21,623**	**$61,973**	**$733,502**

Week Number

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15

Location

Best Sellers

Outlet	Order Quantity
Birmingham	**2320**
Belgian waffle	778
Crepe Japonnais	879
Tartiflette Galette	663
Cardiff	**1441**
Belgian waffle	414
Crepe Japonnais	514
Tartiflette Galette	513
Edinburgh	**4296**
Belgian waffle	1228
Crepe Japonnais	1635
Tartiflette Galette	1433
Ipswich	**1605**
Belgian waffle	547
Crepe Japonnais	672
Tartiflette Galette	386
London	**1586**
Total	**21396**

Revenue