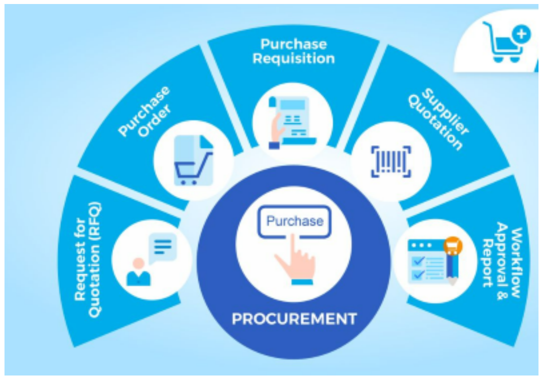
Procurement Department Data Exploration

March 25, 2022

Procurement Department Data Exploration



Contents

Introduction

Over the years,The development of technology and artificial intelligence (AI) has improved the way businesses manage their data over time, as they can now perform data science to make better use of their data. When compared to how business companies used to manage their data prior to data science, a lot of potentially valuable data has been left unused so firms make bad business decisions.

Companies are increasingly attempting to leverage data science by conducting data exploration over their data in order to process, analyze, and drive better business decisions and strategies. They also want to increase revenue and reduce spending by implementing the best recommendations, as this may lead to new business opportunities.

The framework of this project to assist small and large businesses in gaining more valuable insights from their data systems and reduce the costs for this company.

I have been asigned by the procurment to create an EDA for their company "X" and a group of sub-companies located in various Middle-East countries with alot of branches in various fields to assist them in analyzing the correct results based on the business contexts and also in gaining insights that they are unaware of.

We will work on a real data set sample provided by their IT department in this notebook. Data can be found on this link: https://drive.google.com/file/d/1BV-nnfmkhSHzdzCBJjpqh89BD9-5PbuT/view?usp=sharing

Note: **To preserve the confidentiality and privacy of the target company, we will refer to is as X within the report.**

Problem Statement

Purchase order (OP) is an external document issued by the buyer to the seller, usually in response to the offer or proposal. It is a legally binding document created prior to the delivery of services or goods. Whatever the nature of the organization's business, the accuracy of financial figures is critical for all businesses. Accurate financial data are critical because management, stockholders and external auditors rely on comprehensive and reliable information to assess the company's financial status and performance and ensure that costs remain within budget.

Because of that, company X should be aware of all relevant business information specifics based on their industry. Due to the nature of this company's business, the finance department noticed that the cost of the company's purchase was quite high and their problem was a lack of control over their purchases. Furthermore, in some cases, the procurement process will most likely fail to perform as planned in the near future. As a result, they think that their financial data is insufficiently accurate and they've decided to look into these costs, as well as trying to define and prevent future problems.

As a data scientist have been assigned this task, our main goal is to smooth out and clarify the purchase order process, Moreover, when they detects from insights that costs are about to exceed the budget, they can take the necessary actions, such as increasing cost forecasts or avoiding enormous costs and this goal can be achieved by conducting exploratory data analysis and visualizing all of the analyses, as well as answering some critical process questions.

Part 1: Data Collection

Step 1: Install Modules

Some of the following libraries that will be used in the data exploration must be installed as a prerequisite for the code to work properly.

To do so, run **pip install**

1. **`Missingno Library`:** To gain a better understanding of the distribution of NAN values, use informative and detailed visualizations of missing data.
2. **`SideTable Library`:** Allows for the use of a variety of useful methods for exploratory data analysis on the dataset.

[1]:
```
pip install missingno
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: missingno in
```

```
/home/rashaalbadarneh/.local/lib/python3.8/site-packages (0.5.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.8/dist-packages
(from missingno) (1.19.5)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.8/dist-
packages (from missingno) (3.4.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.8/dist-packages
(from missingno) (1.7.1)
Requirement already satisfied: seaborn in /usr/local/lib/python3.8/dist-packages
(from missingno) (0.11.1)
Requirement already satisfied: python-dateutil>=2.7 in
/usr/local/lib/python3.8/dist-packages (from matplotlib->missingno) (2.8.2)
Requirement already satisfied: pyparsing>=2.2.1 in
/usr/local/lib/python3.8/dist-packages (from matplotlib->missingno) (2.4.7)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.8/dist-
packages (from matplotlib->missingno) (8.3.1)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.8/dist-packages (from matplotlib->missingno) (1.3.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.8/dist-
packages (from matplotlib->missingno) (0.10.0)
Requirement already satisfied: pandas>=0.23 in /usr/local/lib/python3.8/dist-
packages (from seaborn->missingno) (1.3.1)
Requirement already satisfied: six in /usr/local/lib/python3.8/dist-packages
(from cycler>=0.10->matplotlib->missingno) (1.15.0)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.8/dist-
packages (from pandas>=0.23->seaborn->missingno) (2021.1)
Note: you may need to restart the kernel to use updated packages.
```

[2]:
```
pip install sidetable
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: sidetable in
/home/rashaalbadarneh/.local/lib/python3.8/site-packages (0.9.0)
Requirement already satisfied: pandas>=1.0 in /usr/local/lib/python3.8/dist-
packages (from sidetable) (1.3.1)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.8/dist-
packages (from pandas>=1.0->sidetable) (1.19.5)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.8/dist-
packages (from pandas>=1.0->sidetable) (2021.1)
Requirement already satisfied: python-dateutil>=2.7.3 in
/usr/local/lib/python3.8/dist-packages (from pandas>=1.0->sidetable) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.8/dist-
packages (from python-dateutil>=2.7.3->pandas>=1.0->sidetable) (1.15.0)
Note: you may need to restart the kernel to use updated packages.
```

Step 2: Import important libraries

To start practicing exploratory data analysis, look at and describe the data set from different angles. Tools, useful functions and methods should be imported to make the EDA process seamless.

Some of the most important libraries:

1. `Pandas` : This will help with data analysis and manipulation.

2. `numpy` : This helps in performing a variety of mathematical operations on numerical arrays.

3. `Sklearn`: This will have an impact on our machine learning models by allowing us to use different functions within the model.

4. `matplotlib, seaborn`: Display our results.

5. `mpl_toolkits`: For better **3D** visualization.

6. `Sidetable`: To aid in the creation of summary tables in Pandas in order to best summarize the results.

7. `Itertools Module`: Importing the combinations library to efficiently use itertools' different subfunctions will aid in efficient looping with better performance in time and space.

8. `Collection Module`: Importing the Counter library makes it easier to perform arithmetic operations and count values because they will be stored in a key-value pair.

9. `Warning`: To control warnings, use the simplefilter function and pass the action that needs to be performed in the next warning by passing ignore.

[3]:
```python
import pandas as pd
import numpy as np
import seaborn as sns
import datetime

import missingno as msno
import sidetable as stb
import warnings
import operator

import matplotlib.pyplot as plt
import plotly.graph_objects as go

from matplotlib import style
from itertools import combinations
from collections import Counter

warnings.simplefilter(action='ignore')
```

Step 3: Load the data

A **CSV file** is the most common file type to store data.

By using the **read csv()** function and passing the directory of the dataset file, it can ba easily access the dataset and understand its structure.

Below, basic operations will be performed to check what the data consists of.

- `Head of the dataset.`

- `Shape of the dataset.`
- `info of the dataset.`
- `Summary of the overall dataset.`

Once the data is loaded into a data frame, there is a need to set an index on the dataset by one of the data columns.

```
[10]: df = pd.read_csv("../../datasets/purchased_orders.csv", thousands=",")
```

Start exploring dataset and its features by printing the first five rows from our data frame to have a general view of the selected dataset by performing the following command.

```
[11]: df.head()
```

```
[11]:    sequence_no  company_code company_name order_no order_type business_unit  \
      0            1           200       Jordan   536365         OP        166552
      1            2           200       Jordan   536365         OQ           M30
      2            3           200       Jordan   536365         O4           M30
      3            4           200       Jordan   536365         O4           M30
      4            5         16655       Turkey   536365         OP        166552

         supplier_no request_date order_date  item_no  \
      0       163897    2/11/2022  2/11/2022   760350
      1         4343     2/1/2022   2/1/2022    60011
      2         4343    1/31/2022  1/31/2022   700023
      3         4343    1/31/2022  1/31/2022   700023
      4       163897    2/11/2022  2/11/2022   760350

                          description line_type UOM   quantity_ordered  \
      0  Raw Materail purchase for HYD         S    EA             1000.0
      1              Mountain Bike, Red         S    EA             1234.0
      2            Multivitamin Tablets         S    PC                5.0
      3            Multivitamin Tablets         S    PC                3.0
      4  Raw Materail purchase for HYD         S    EA             1000.0

         quantity_open  quantity_received  unit_cost  extended_cost Item_ctg  FY
      0         1000.0                1.0   100.0000      100000.00     IN30   0
      1         1234.0                1.0   530.1000      654143.40     IN30   0
      2            NaN                1.0     0.3423           1.71     IN30   0
      3            3.0                1.0     0.3423           1.03     IN30   0
      4         1000.0                1.0   100.0000      100000.00     IN30   0
```

The correct data, as shown above, is from index number 1. As a result, we must correct the first two rows by changing the data frame's index to **'sequence no'**.

```
[12]: df.set_index("sequence_no")
```

```
[12]:             company_code  company_name order_no order_type business_unit  \
sequence_no
1                        200        Jordan   536365         OP        166552
2                        200        Jordan   536365         OQ           M30
3                        200        Jordan   536365         O4           M30
4                        200        Jordan   536365         O4           M30
5                      16655        Turkey   536365         OP        166552
…                        …            …        …            …           …
1745                       1  Saudi Arabia   536544         OP           M30
1746                       1  Saudi Arabia   536544         OP           M30
1747                       1  Saudi Arabia   536544         OP           M30
1748                       1  Saudi Arabia   536544         OP           M30
1749                       1  Saudi Arabia   655654         OP           M30

            supplier_no request_date order_date  item_no  \
sequence_no
1                163897    2/11/2022  2/11/2022   760350
2                  4343     2/1/2022   2/1/2022    60011
3                  4343    1/31/2022  1/31/2022   700023
4                  4343    1/31/2022  1/31/2022   700023
5                163897    2/11/2022  2/11/2022   760350
…                   …           …          …       …
1745               4343     6/5/2017   6/5/2017  5483902
1746               4343     6/5/2017   6/5/2017  5483902
1747               4343     6/5/2017   6/5/2017  5483902
1748               4343     6/5/2017   6/5/2017  5483902
1749               4343     6/5/2017   6/5/2017  5483902

                          description line_type UOM   quantity_ordered  \
sequence_no
1            Raw Materail purchase for HYD         S   EA            1000.0
2                       Mountain Bike, Red         S   EA            1234.0
3                     Multivitamin Tablets         S   PC               5.0
4                     Multivitamin Tablets         S   PC               3.0
5            Raw Materail purchase for HYD         S   EA            1000.0
…                                        …         …   …                  …
1745                           Spare Parts         S   EA             250.0
1746                           Spare Parts         S   EA             250.0
1747                           Spare Parts         S   EA             250.0
1748                           Spare Parts         S   EA             250.0
1749                           Spare Parts         S   EA             250.0

            quantity_open  quantity_received  unit_cost  extended_cost  \
sequence_no
1                  1000.0                1.0   100.0000      100000.00
2                  1234.0                1.0   530.1000      654143.40
3                     NaN                1.0     0.3423           1.71
```

```
4                     3.0                 1.0     0.3423          1.03
5                  1000.0                 1.0   100.0000      100000.00
…                       …                   …          …              …
1745                  0.0               250.0    10.0000        2500.00
1746                  0.0               250.0    10.0000        2500.00
1747                  0.0               250.0    10.0000        2500.00
1748                  0.0               250.0    10.0000        2500.00
1749                  0.0               250.0    10.0000        2500.00

            Item_ctg  FY
sequence_no
1               IN30   0
2               IN30   0
3               IN30   0
4               IN30   0
5               IN30   0
…                …   …
1745            IN40  17
1746            IN40  17
1747            IN40  17
1748            IN40  17
1749            IN40  17

[1749 rows x 19 columns]
```

As it's observed, there are a bunch of different records with 19 features that will be used in our EDA process.

Now, check the last five rows of the dataset by using the following command.

```
[13]: df.tail(5)
```

```
[13]:      sequence_no  company_code  company_name order_no order_type  \
1744         1745             1  Saudi Arabia   536544         OP
1745         1746             1  Saudi Arabia   536544         OP
1746         1747             1  Saudi Arabia   536544         OP
1747         1748             1  Saudi Arabia   536544         OP
1748         1749             1  Saudi Arabia   655654         OP

     business_unit  supplier_no request_date order_date  item_no  description  \
1744           M30         4343     6/5/2017   6/5/2017  5483902  Spare Parts
1745           M30         4343     6/5/2017   6/5/2017  5483902  Spare Parts
1746           M30         4343     6/5/2017   6/5/2017  5483902  Spare Parts
1747           M30         4343     6/5/2017   6/5/2017  5483902  Spare Parts
1748           M30         4343     6/5/2017   6/5/2017  5483902  Spare Parts

     line_type UOM  quantity_ordered  quantity_open  quantity_received  \
1744         S  EA             250.0            0.0              250.0
```

```
1745         S   EA             250.0            0.0              250.0
1746         S   EA             250.0            0.0              250.0
1747         S   EA             250.0            0.0              250.0
1748         S   EA             250.0            0.0              250.0

      unit_cost  extended_cost Item_ctg  FY
1744       10.0         2500.0     IN40  17
1745       10.0         2500.0     IN40  17
1746       10.0         2500.0     IN40  17
1747       10.0         2500.0     IN40  17
1748       10.0         2500.0     IN40  17
```

Step Four: Dataset Dimension

Now, check for the dataset dimension and the number of features and attributes that exist by using `shape` function to get the number of rows and columns.

[14]:
```
print("Dataset Rows:", df.shape[0])
print("Dataset Columns:", df.shape[1])
```

```
Dataset Rows: 1749
Dataset Columns: 20
```

Insights:

The dataset has a total of 1749 observation rows and a total of 20 variable columns with different data types.

[15]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1749 entries, 0 to 1748
Data columns (total 20 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   sequence_no       1749 non-null   int64
 1   company_code      1749 non-null   int64
 2   company_name      1749 non-null   object
 3   order_no          1749 non-null   object
 4   order_type        1749 non-null   object
 5   business_unit     1749 non-null   object
 6   supplier_no       1749 non-null   int64
 7   request_date      1748 non-null   object
 8   order_date        1749 non-null   object
 9   item_no           1749 non-null   int64
 10  description       1749 non-null   object
 11  line_type         1749 non-null   object
 12  UOM               1749 non-null   object
 13  quantity_ordered  1705 non-null   float64
 14  quantity_open      1055 non-null   float64
 15  quantity_received  1346 non-null   float64
 16  unit_cost          1749 non-null   float64
 17  extended_cost      1298 non-null   float64
 18  Item_ctg           1749 non-null   object
 19  FY                 1749 non-null   int64
dtypes: float64(5), int64(5), object(10)
memory usage: 273.4+ KB
```

From the above results, we have the following metadata description for our dataset:

- The dataset has___10 numeric___values and___10 categorical___values.
- Independent variables are divided into ”categorical” and ”numerical” variables as well.
- Numerical variables in the data set include: ['Unit Cost,' 'Item No,' 'Order No,' 'Supplier No,' and 'Extended Cost.']
- Categorical variables in the data set includes: ['Item Category', 'Line Type', 'Description', 'Company name', 'Unit of Measure]
- Some columns have missing values (This fact will be examined in further steps).
- Some columns do not have the correct data types; therefore, changes must be made to these columns.

We can't assume that all the data was loaded into the correct data types, so types will be used to check the data types for all the data. So let us get the data type of each column in the data frame by using the **df.dtypes** attribute.

[16]:
```
df.dtypes
```

[16]:
```
sequence_no            int64
company_code           int64
company_name          object
order_no              object
order_type            object
business_unit         object
supplier_no            int64
request_date          object
order_date            object
item_no                int64
description           object
line_type             object
UOM                   object
quantity_ordered     float64
quantity_open        float64
quantity_received    float64
unit_cost            float64
extended_cost        float64
Item_ctg              object
```

```
FY                     int64
dtype: object
```

Looking at the data frame

request date and **order date** are of **DateTime** datatype rather than object type.

Description of dataframe attributes:

The following table represents the features of the data frame.

Attribute Name	Description
company_code	Unique identifier as a reference number for each company in different countries.
company_name	The name of company
order_no	Order Number
order_type	The type of purchase order
business_unit	Costs center for every company
supplier_no	Suppliers numbers
request_date	The delivery date for the requested order
order_date	Date for each PO order

Attribute Name	Description
item_no	Number identify each item
description	Order name and it's description
line_type	Item type(stock or non-stock items)
UOM	Unit of measure
quantity_ordered	The quantity of each purchase order per transaction.
quantity_open	Ordered quantity but have not yet recieved
quantity_received	Ordered quantity and recieved
unit_cost	Cost per unit for item
extended_cost	Total cost
Item_ctg	The item's category
FY	fiscal year

[17]:
```
print(
    "Number of Categorical data in the dataset is: ",
    df.select_dtypes(include=["object"]).dtypes.count(),
)

print(
    "Number of Numerical data in the dataset is: ",
    df.select_dtypes(include=["int", "float"]).dtypes.count(),
)
```

```
Number of Categorical data in the dataset is:  10
Number of Numerical data in the dataset is:  10
```

Step Five: Get Dataset copy

Before starting the data preprocessing on the dataset, it's better to create a `copy of the original data frame` that will help protect the initial data frame and indices from being changed or manipulated elsewhere.

[18]:
```
dataset_copy = df.copy()
```

Part 2: Data Exploration

Step 1: Data Inspection

In this part, some techniques will be used to drive deep into the dataset in order to describe the characterizations of the dataset, such as missing values, unique values, duplicates and outliers to strategize and identify the relationships between the target variable and different variables, define the distribution of the variables for a better understanding of the data and gaining better insights to effectively build a regressor model that fits the data.

`1.1 Null Values`

Missing values can cause problems and errors and directly affect what we need to do in the data cleaning step as well as affect the final insights. Therefore, in this section, the main step is to check for any missing (NULL) values to decide what actions will be taken on them if there are any. However, as the sidetable library has been imported, it would be much better to add some attributes that help build a simple missing values table by using the function missing that provides more information about the missing values and the percentage of each missing value in each column.

This will be much easier to interpret and more comprehensible when it comes to quickly identifying the details of missing values, which leads to having a clear reason for the actions that need to be performed to treat the missing values.

Check for any `missing values` separately by using **`isnull()`** functions with **`sum()`** to return the number of missing values.

[19]:
```
df.stb.missing(style=True)
```

[19]:
```
<pandas.io.formats.style.Styler at 0x7f63ebee8370>
```

Insights:

We can see that the respective columns have a variety of missing values.

Columns that have missing values:

`QUANTITY OPEN` has the highest percentage of missing values, followed by `EXTENDED COST` and `QUANTITY RECIEVED`, while `QUANTITY ORDERED` and `REQUEST DATE` have the lowest percentage of missing values.

We will treat these missing values based on the following techniques

1. Remove the rows with NAN values in the columns (QUANTITY OPEN, QUANTITY ORDERED, REQUEST DATE, QUANTITY RECEIVED).

2. Replace the missing values in the EXTENDED COST column with a formula that will be explained later on.

Understanding the level of missing data in dataset analysis should be one of the first things we should focus on when doing data exploration over the dataset and before deciding how to treat missing values. As a result, by using the following formula, we can calculate the overall missing value percentage in the dataset. We'll employ the following formula:

Percentage of NAs = (Number of cells with NA) * 100 /(Total number of cells)

Methods used:

- Determine number of cells in total by taking the product of all the elements in the input array where the product is over the given array shape and storing this value as the total product using the NumPy array because it is faster and more efficient.
- Calculate the NA cells by using **isna()** and appending **.sum()** to get the null values in the data frame and calculating the sum of these NA values, then assigning the result in the object count to use in the formula.
- Print and format the result of the Percentage of NANs formula. The format is as follows:

[20]:
```
total_cells = np.product(df.shape)
count = df.isna().sum().sum()
percent_missing = (count / total_cells) * 100
print(f"{percent_missing:.2f}%")
```

```
4.55%
```

The total percentage of missing data is significant. In general, if fewer than 5% of the values are missing, it is acceptable to ignore them or to delete the NA rows by dropping, deleting columns with NAN values, or imputing the Nan values. All of these decisions are based on the EDA's purpose and the exact result that must be generated by this EDA.

[22]:
```
print("Number of NAN Rows:")
print(df.isnull().any(axis=1).sum())

print("Number of NAN Columns:")
print(df.isnull().any(axis=0).sum())
```

```
Number of NAN Rows:
1274
Number of NAN Columns:
5
```

Missing values are distributed over the dataframe as 1274 rows and 5 columns have missing values.

1.2 Unique Values

It's a good idea for a large company that stores a large volume of datasets to check the unique values in each column and have an idea of these values.

We will be able to filter out only unique values from the categorical columns and define a function to automate checking the unique values across the columns with the datatype object by using Pandas' **unique() function**

[27]:
```
categorical_unique = df.select_dtypes(["object"]).columns

for col in categorical_unique:
    print("{} : {} unique value(s)".format(col, df[col].nunique()))
```

```
company_name : 8 unique value(s)
order_no : 91 unique value(s)
order_type : 10 unique value(s)
business_unit : 28 unique value(s)
request_date : 155 unique value(s)
order_date : 154 unique value(s)
description : 246 unique value(s)
line_type : 9 unique value(s)
UOM  : 18 unique value(s)
Item_ctg : 11 unique value(s)
```

Observing the highest number of categorical unique values in the **description** column with **246 unique values** let's explore the unique values by defining the most unique values column and using unique() as follows.

[32]:
```
df["description"].unique
```

[32]:
```
<bound method Series.unique of 0       Raw Materail purchase for HYD
1                    Mountain Bike, Red
2                  Multivitamin Tablets
3                  Multivitamin Tablets
4        Raw Materail purchase for HYD
                       …
```

```
1744                          Spare Parts
1745                          Spare Parts
1746                          Spare Parts
1747                          Spare Parts
1748                          Spare Parts
Name: description, Length: 1749, dtype: object>
```

Same step for numerical data.

```
[33]: numerical_data = df.select_dtypes(["int", "float"]).columns
```

```
[34]: for col in numerical_data:
          print("{} : {} unique value(s)".format(col, df[col].nunique()))
```

```
sequence_no : 1747 unique value(s)
company_code : 8 unique value(s)
supplier_no : 62 unique value(s)
item_no : 440 unique value(s)
quantity_ordered : 121 unique value(s)
quantity_open : 97 unique value(s)
quantity_received : 90 unique value(s)
unit_cost : 173 unique value(s)
extended_cost : 304 unique value(s)
FY  : 11 unique value(s)
```

```
[35]: df["item_no"].unique
```

```
[35]: <bound method Series.unique of 0        760350
1         60011
2        700023
3        700023
4        760350
          …
1744    5483902
1745    5483902
1746    5483902
1747    5483902
1748    5483902
Name: item_no, Length: 1749, dtype: int64>
```

1.3 Duplicates

Duplicate values in a dataset must be identified and handled because having the same records in the dataset will not help the company make the best decisions and insights. So, in this step, it's important to check for the number of duplicates and remove any duplicates by deciding to keep only the first, last record or to keep none and drop all the duplicates.

As shown below, count the number of duplicated values. * To count the number of duplicated rows in a pandas data frame using **duplicated()**

[36]:
```
print("Total No. of duplicated rows in our dataset: {}".format(df.duplicated().
↪sum()))
```

```
Total No. of duplicated rows in our dataset: 2
```

The dataset only has two duplicates, which is a manageable number. For further investigation, we will use Pandas' duplicated method to display the duplicated rows in the dataset.

[37]:
```
df[df.duplicated()]
```

[37]:
```
      sequence_no  company_code  company_name order_no order_type  \
1485         1485             1  Saudi Arabia   536544         OP
1663         1662           200        Jordan   536544         OP

     business_unit  supplier_no request_date  order_date  item_no  \
1485        166552       163897    2/11/2022   2/11/2022   760350
1663            30       533103   12/27/2017  12/27/2017   740068

                          description line_type UOM  quantity_ordered  \
1485  Raw Materail purchase for HYD         S   EA             100.0
1663                      Desk - Oak         S   EA            4000.0

      quantity_open  quantity_received  unit_cost  extended_cost Item_ctg  FY
1485          100.0              100.0      100.0        10000.0     IN30   0
1663            0.0             4000.0      950.0          950.0     IN20  17
```

Using Pandas' **describe** function, display a statistical summary for the dataset and round the numbers as follows.

[38]:
```
df.describe().T.round(2)
```

[38]:
```
                    count       mean         std       min       25%  \
sequence_no        1749.0     874.86      504.83      1.00    438.00
company_code       1749.0     225.47     2147.16      1.00      1.00
supplier_no        1749.0   43357.74   126929.76    500.00   4343.00
item_no            1749.0  996909.33  1522320.52  60003.00  60839.00
quantity_ordered   1705.0    1000.38     9616.41     -3.47      0.08
quantity_open      1055.0     895.32     9882.30     -3.47      0.04
quantity_received  1346.0     563.14     5330.89      0.01      0.30
unit_cost          1749.0  119107.38  1513361.59   -240.00      5.00
extended_cost      1298.0  140367.44  3667566.24      0.08     92.25
FY                 1749.0      10.37        9.74      0.00      0.00

                        50%       75%        max
sequence_no           875.0    1312.0     1749.0
company_code            1.0     200.0    48264.0
supplier_no          4343.0    9292.0   533104.0
item_no            736587.0  740674.0  5567889.0
```

```
quantity_ordered         1.0       30.0      234567.0
quantity_open            1.0       23.0      234567.0
quantity_received        1.0        1.0      100000.0
unit_cost               15.0       68.5    20000000.0
extended_cost          625.0     3000.0   124343966.7
FY                      16.0       17.0          23.0
```

As the statistical summary of the dataset shows some statistical data for numerical columns such as (percentile, mean, max, min, and standard deviation), we can easily observed that there are some outliers identified in some columns where there is a difference of 75% between the maximum value and the third quartile.

Following this numerical description, questions may asked.

- **Having NEGATIVE values**:

Having negative values in the dataset, means that the item is returned to the supplier or being removed from the company stock.

Regarding the 'order_no' column, that should be an int value, whereas it is an object value, use value_counts() to explore the values and have some insights.

[64]:
```
df["order_no"].value_counts().to_frame()
```

[64]:
```
            order_no
536544           305
536464            85
536412            81
536532            73
536520            71
…                …
536414             1
536380             1
C536379            1
C536383            1
655654             1

[91 rows x 1 columns]
```

It's been observed that there are some values begin with letter C withing the number. Explore more details about these values by performing the follwoing command.

[44]:
```
c_orders = df["order_no"].str.startswith("C").value_counts()
print("\n Orders that start with letter C:")
print(c_orders)
```

```

 Orders that start with letter C:
False    1737
```

```
True       12
Name: order_no, dtype: int64
```

This column, as shown, contains 12 values that begin with the **letter C**. As a result, we can be sure that these negative values in the quantity column correspond to each canceled order number.

1.4 Outliers

Outliers are data points that differ significantly from the remaining data points in the dataset. They can appear for a variety of reasons, such as errors made while measuring or entering data. However, in some cases, outliers must be identified and handled, whereas in others, they are not important and can be ignored.

based on our data set

To find outliers in the dataframe, do the following:

- **Visualise the outliers.**
- **Define a function to find the outliers in the data frame.**
- **Use some strategies to deal with outliers, either by eliminating them or accepting them.**

Begin by defining the important numerical columns to be checked and represented using the **seaborn** library, which aids in detecting outliers clearly.

Using the following techniques, visualize the outliers:

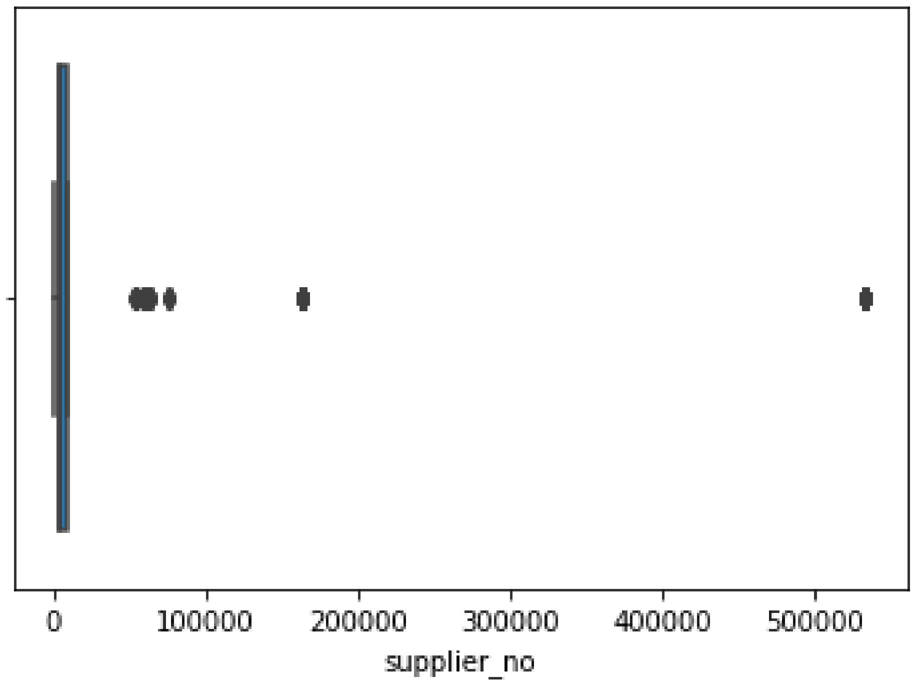
1. Create an array of quantitative data that will be checked for outliers.

2. Use the 'boxplot' function from the'seaborn' library to visualize the distribution of points in numerical columns, passing each column by its index.

[45]:
```
boxplotcolumns = ["supplier_no", "extended_cost", "item_no"]
```

[46]:
```
sns.boxplot(df[boxplotcolumns[0]])
```

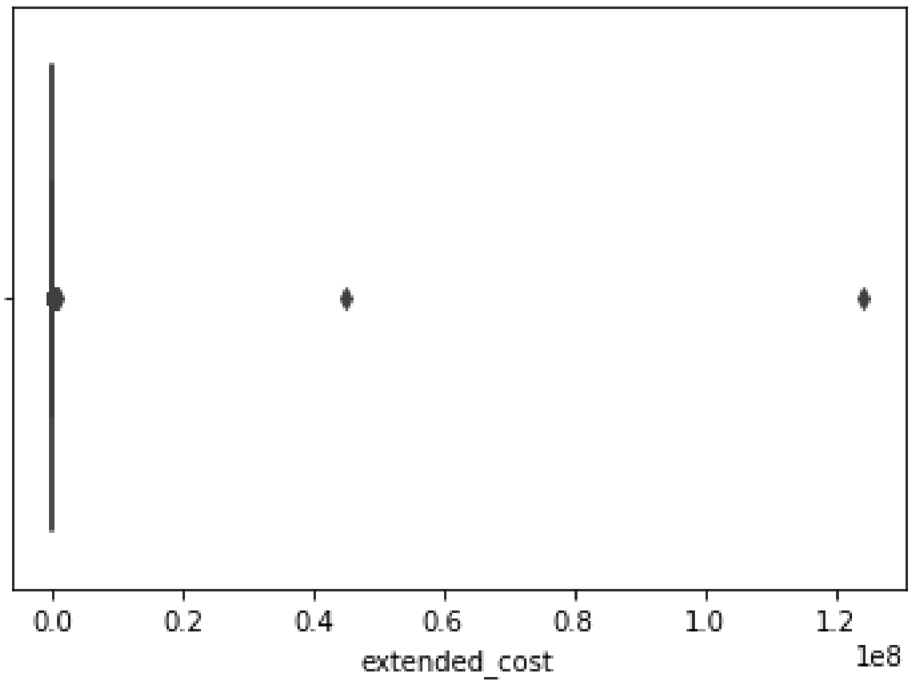
[46]:
```
<AxesSubplot:xlabel='supplier_no'>
```



As can be seen, supplier number contains some maximum outliers where we can consider them as the extreme values.

[47]:
```
sns.boxplot(df[boxplotcolumns[1]])
```

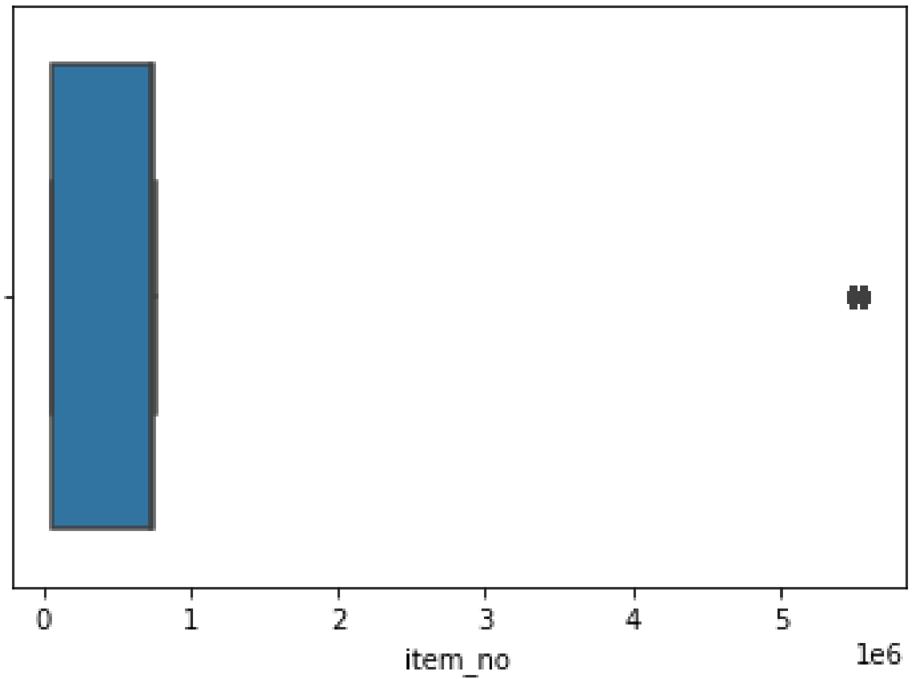
[47]:
```
<AxesSubplot:xlabel='extended_cost'>
```



Extended cost cloumn has a few maximum outliers.

[48]:
```
sns.boxplot(df[boxplotcolumns[2]])
```

[48]:
```
<AxesSubplot:xlabel='item_no'>
```



For item number, only two outliers can be observed.

To gain a clearer understanding of the outliers in the dataset, we will define a function to find and return a list of outlier indexes in a specific column, and we will handle them by defining a function to remove the unwanted outliers (in the data preprocessing part) if this will be the best decision.

To identify the outliers in the dataframe, by defining the function find outlier with one input argument x to do the following:

- Select **first quartile Q1** and **third quartile Q3** for each column by defining the variables Q1 and Q3 and set the their percentiles.
- Compute the interquartile range **(IQR)** where it equals the third quartile-first quartile ('IQR = Q3 - Q1'), the range of normal data.
- Print the names of the columns that were passed to the function.
- Define the counter for the number of outliers(that fall above the Q3 as maximum outliers OR below Q1 as minimum) in these columns by access these column by label using **loc**.

[54]:
```
for cols in boxplotcolumns:
    Q3 = df[cols].quantile(0.75)
    Q1 = df[cols].quantile(0.25)
    IQR = Q3 - Q1

    print(f"{cols.capitalize()} column")
    count = len(df.loc[(df[cols] < (Q1 - 1.5 * IQR)) | (df[cols] > (Q3 + 1.5 *␣
↪IQR))])
    print(f"no of records with outliers values: {count}")
```

```
Supplier_no column
no of records with outliers values: 215
Extended_cost column
no of records with outliers values: 182
Item_no column
no of records with outliers values: 171
```

- We can conclude that in the supplier number column, it has an amount of 215 outliers
- For extended cost column, ther is 182 outliers and the item_no column contains of 171 outliers.

Step 2: Visualizing data points and features

In this section, we'll show some graphs and plots to gain some key insights and correlations.

Let us begin by going deeper into the variables, examining the relationships between them as well as the relationships between the variables.

`Correlation`

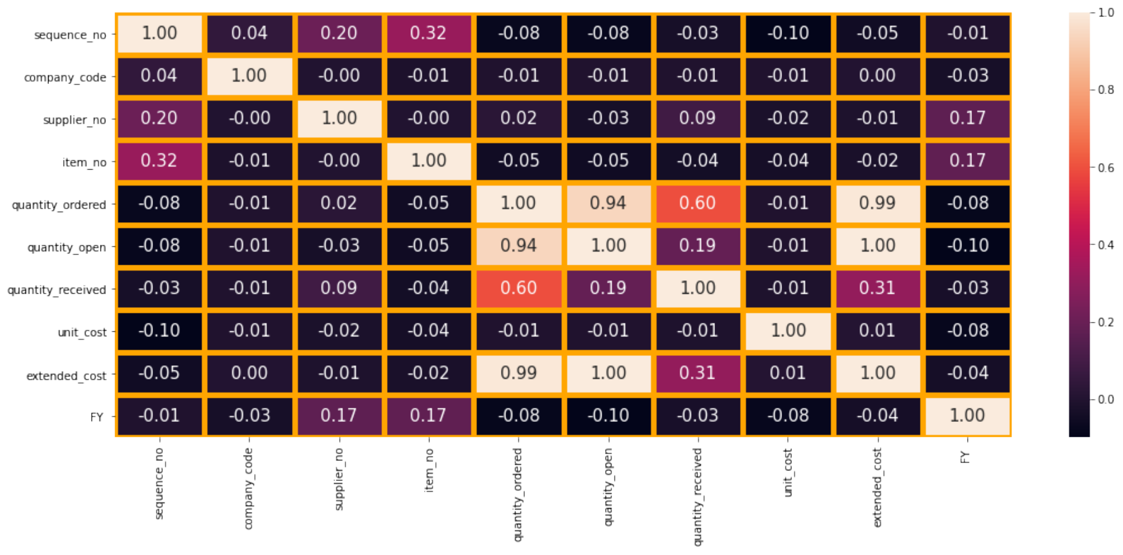
Plot the results using the'seaborn's heatmap' function and the parameters:

- Correlation data.
- `annot`: To write the correlation data value in each cell, set is to True.
- `fmt`: This function is used to format the string that will be used to write annotation values.
- `annot_kws`: Set the size to have more clear vision of the matrix.
- `linewidth`: the width of the dividing line between cells.
- `line color`: to choose the line color.

Checking the correlation between variables in the dataset is also required to determine the most significant values for further analysis and answer the business questions later.

To accomplish this, we will create a correlation heatmap by specifying the matrix size as well as all other properties for better visualization.

[57]:
```
plt.figure(figsize=(18, 7))
sns.heatmap(
    df.corr(),
    annot=True,
    fmt="0.2f",
    annot_kws={"size": 15},
    linewidth=5,
    linecolor="orange",
)
plt.show()
```



Observations

Features are highly correlated with each other and are dependent where the darker light indicate a low correlation and the lighter light indicate a high correlation.

Positive Correlation:

- Extended cost has a positive and high relationship with the quantity ordered as well as with the quantity open.
- Quantity ordered, Quantity received have a high correlation
- Quantity open and quantity ordered have a positive high relationship.
- The quantity received and quantity opened have a low correlation with the supplier number.

Negative Correlation:

- Unit cost and quantity ordered have a negative correlation.
- Supplier number and quantity open

Part 3: Data Preprocessing

Step 1: Data cleaning

In this step, we'll look for the following to make changes to the dataset in order to get the best insights and make it easier to answer business question at the end.

- Check for column names' corrections.
- Handle missing values.
- Handle duplicate values.
- Include new features as well as a column.
- Remove any unnecessary columns.
- Perform any additional steps.

`1.1 Modify data type`

A column representing dates is categorical data and should have the data type **Datetime**.

To obtain the date and time conversion, run the following commands.

[58]:
```python
df["request_date"] = pd.to_datetime(df["request_date"])

df["order_date"] = pd.to_datetime(df["order_date"])
```

[59]:
```python
df.dtypes.value_counts()
```

```
[59]: object            8
      int64             5
      float64           5
      datetime64[ns]    2
      dtype: int64
```

Successfully converted.

To gain a better understanding of the data and access the columns easily, check for the names of categorical and numerical columns.

```
[60]: df.select_dtypes(include=["object"]).columns.tolist()
```

```
[60]: ['company_name',
       'order_no',
       'order_type',
       'business_unit',
       'description',
       'line_type',
       'UOM ',
       'Item_ctg']
```

```
[61]: df.select_dtypes(include=["int", "float"]).columns.tolist()
```

```
[61]: ['sequence_no',
       'company_code',
       'supplier_no',
       'item_no',
       'quantity_ordered',
       'quantity_open',
       'quantity_received',
       'unit_cost',
       'extended_cost',
       'FY ']
```

as mentioned, the order number column includes the letter ”C” alongside the numbers, indicating that they are canceled orders. As a result, the column contains the correct data type and no conversion is required.

1.2 Rename columns

The following process will be followed:

* Create a 'dictionary object' with two values: the old name (the dictionary's **key**) and the new name (the key's **value**).

- Change 'FY' thas has a space in the name to 'fiscal year'.
- 'UOM ' to 'unit_of_measue'

```
[65]: new_name = {"FY ": "fiscal_year", "UOM ": "unit_of_measure"}
```

Now, Rename the columns in the data frame.

```
[66]: df = df.rename(columns=new_name)
      df.columns
```

```
[66]: Index(['sequence_no', 'company_code', 'company_name', 'order_no', 'order_type',
             'business_unit', 'supplier_no', 'request_date', 'order_date', 'item_no',
             'description', 'line_type', 'unit_of_measure', 'quantity_ordered',
             'quantity_open', 'quantity_received', 'unit_cost', 'extended_cost',
             'Item_ctg', 'fiscal_year'],
            dtype='object')
```

The columns are clearly named correctly, as can be seen.

1.3 Drop unnecessary columns

Drop any superfluous columns that will not be used in the analysis process. Using drop and pass the column name, and the axis 1 to represent for a column and modify the orginal dataframe inplace by set it to TRUE.

```
[67]: df.drop("fiscal_year", inplace=True, axis=1)
```

```
[69]: df.drop("business_unit", inplace=True, axis=1)
```

1.4 Add new columns

We'll split the order date column into two parts so that the year of the purchased order can be examined separately, then insert it into our data frame using the **insert()** method, passing the 'dt.year' attribute to return the year of DateTime, and passing the position of the inserted column bypassing 'loc' and the index for the new column.

```
[71]: df.insert(loc=9, column="year_of_order", value=df["order_date"].dt.year)
```

To ensure that the added column is correct, double-check the shape of the data frame.

```
[72]: print("Data Diminsion is:", df.shape)
```

```
Data Diminsion is: (1749, 19)
```

```
[73]: df.columns
```

```
[73]: Index(['sequence_no', 'company_code', 'company_name', 'order_no', 'order_type',
             'supplier_no', 'request_date', 'order_date', 'item_no', 'year_of_order',
             'description', 'line_type', 'unit_of_measure', 'quantity_ordered',
             'quantity_open', 'quantity_received', 'unit_cost', 'extended_cost',
             'Item_ctg'],
            dtype='object')
```

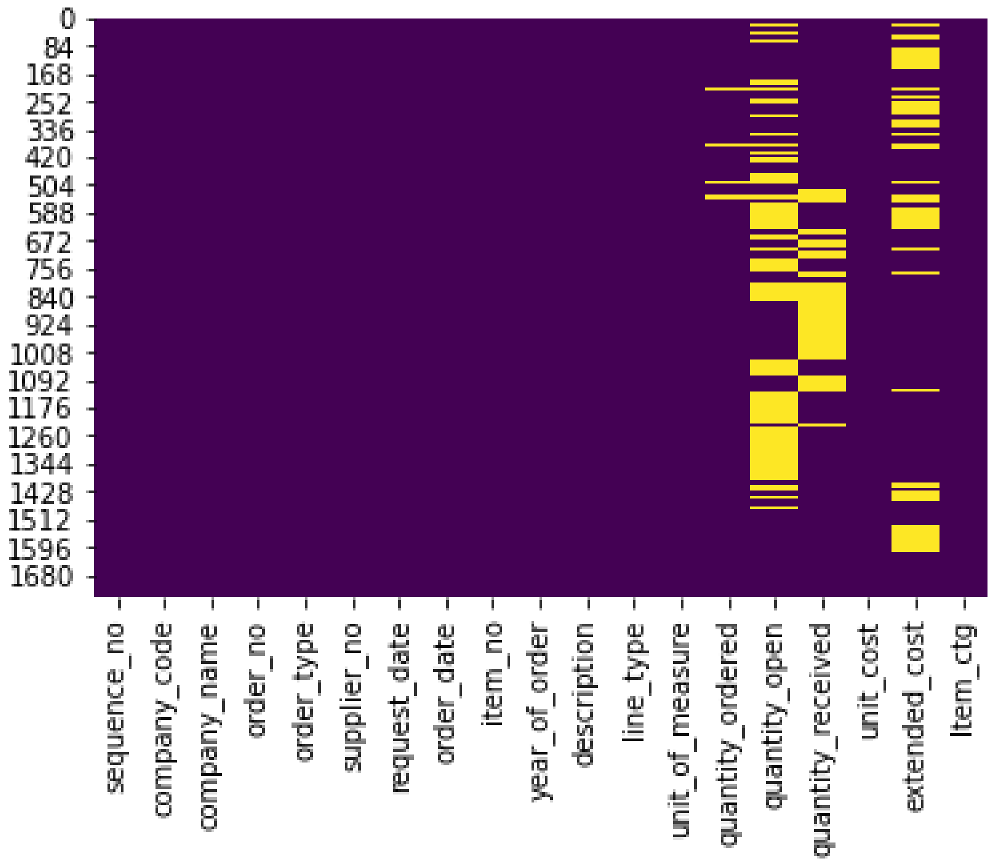
We can see that the columns have increased by one, and it now appears in the column names as the 9th index and 10th column.

1.4 Handle missing values

We'll now utilize the **python's Seaborn library** and the 'heatmap' function to quickly visualize how much data is missing in a data set in a graphical and interactive manner and pass the following: * `cbar`: Whether or not to draw a colorbar to the right of the heatmap. * `cmap`: to customize the heatmap colors.

```
[76]: sns.heatmap(df.isnull(), cbar=False, cmap="viridis")
```

```
[76]: <AxesSubplot:>
```



sns.heatmap displays the following:

The values that are missing are highlighted in yellow:

- When compared to the other columns, the **Quantity Received** column has a few missing numbers.
- The frequency of missing values in the **Extended Cost** column varies.
- The **Quantity Open** column is nearly empty, with some variation in the number of missing data.

After inspecting the extent of the missing data and its percentage before, some decisions have been made to handle the missing values.

1. **For Extended Cost column:** As we can confirm that both values that will be replaced in the EXTENDED COST column are float values, we can make an imputation to the **'Extended_Cost'** NAN values with the formula.

   ```
   **(Quantity Order * Unit Cost)**
   ```

2. **For additional missing value columns:** /ins>

- **Drop** 'Request_Date' rows with NaN values because there is only one Nan value.
- Drop the rows with NAN values in the 'Quantity Opened', 'Quantity Received', and 'Quantity Ordered' columns because our analytical procedure is based on delivered quantities, not open amounts.

Rows that have missing values should be eliminated. To delete the desired rows from the columns with missing values, use the 'dropna' command.

[77]:
```
df.dropna(subset=["request_date"], inplace=True)
```

[78]:
```
df = df.dropna(
    axis=0, subset=["quantity_received", "quantity_open", "quantity_received"]
)
```

[79]:
```
df.shape
```

[79]:
```
(744, 19)
```

Doble check how much the number of missing values has decreased.

[81]:
```
print("Number of missing values:", df.isnull().sum().sum())
```

```
Number of missing values: 269
```

There is still some missing values where they are reduced to 269 so let us explore the missing values ration by following these steps:

- Creating an object that contains the number of missing values in the dataframe.
- Create a table of missing values that includes the counter and ratio for each value.

We need to **concat** two columns and use the counter object to obtain the missing ratio using the techniques below:

1. Rename the missing_count object and have it's result.

2. Calculate the missing ratio for each missing column in the dataset in comparison to all of the rows by using the **div()** function then pass the number of rows using 'len' and the data frame.

3. After having the the table with the number of missing values for all the dataframe columns and it's ration, compare the data frame to the created table with the **ne()** function based on the non missing cloumns and return the columns that have missing values with the number of missing values in the cloumn and the ratio as follows.

```
[82]: missing_count = df.isna().sum()
      table_of_missing = pd.concat(
          [
              missing_count.rename("missing count"),
              missing_count.div(len(df)).rename("missing_ratio"),
          ],
          axis=1,
      ).loc[missing_count.ne(0)]
      table_of_missing
```

```
[82]:                missing count  missing_ratio
      extended_cost            269       0.361559
```

We can clearly see that there is just one column left with 269 missing values, resulting in a ratio of 0.361.

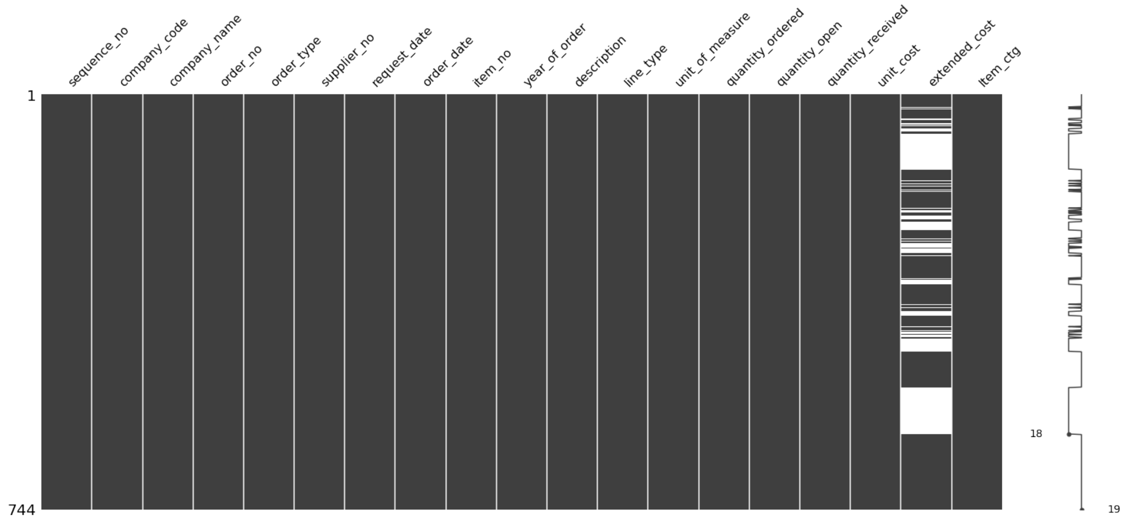
The **missingno** module will be beneficial to utilize and view the distribution of missing data as we progress.

```
[83]: df.shape
```

```
[83]: (744, 19)
```

```
[84]: msno.matrix(df)
```

```
[84]: <AxesSubplot:>
```



Because the grey color is shaded to indicate the present values in the dataset, the white color indicates missing values in the color fill in each column. On the right side of the matrix, there is a line.

This sparkline illustrates how each row's data completeness is shaped. Because the matrix only reveals missing values for the extended cost column, we can see from the right-hand sparkline that when a row has a value in the column, the line will be at the farthest right. The line, on the other hand, will move to the left as the number of missing values increases and appears within that row.

We can't eliminate the extended cost column because it's so critical for our data exploration and insights, thus it's best to use the mentioned formula to fill in the missing numbers.

[85]:
```
df["extended_cost"] = df["extended_cost"].fillna(
    df["quantity_ordered"] * df["unit_cost"]
)
```

Just to be sure, Create a function that checks the amount of missing values once all of them have been handled.

- Create missing values counter
- Calculate the result in a dictionary with the missing count column and get it's value for this column by index and store it in a dataframe.
- Pass the first 40 rows just to check.
- Print the result.

[87]:
```
def find_missing(data):
    count_missing = df.isnull().sum().values
    return pd.DataFrame(
        data={"missing_count": count_missing}, index=data.columns.values
    )


find_missing(df).head(40)

print("Total number of missing values now:", df.isnull().sum().sum())
```

```
Total number of missing values now: 0
```

The process of checking for missing data and dealing with them is now complete. Moving on to the next procedure, which is to verify the dataset for duplicated values.

1.5 Handle Duplicates

In the dataframe there is duplicates. In this scenario, the method drop_duplicated will be utilized to remove duplicate entries while maintaining the value in the first row by updating the data frame and modifying the data in place.

[88]:
```
df.drop_duplicates(keep="first", inplace=True)
```

Check whether the duplicates have been removed from the data set or not by looking at the number of rows.

[89]:
```
df.shape
```

[89]:
```
(742, 19)
```

As the number of rows were 744, now they are reduced by two rows. So we can be sure that we have removed the duplicated successfully.

`1.5 Handle Outliers`

However, given the large number of samples in our dataset, a small number of outliers, such as those indicated above, is to be expected. These outliers are not attributable to any aberrant conditions or fault data. Based on the dataset and its properties. On the other hand, depending on whether the purchased orders are extraordinarily high, they may be the most extreme observations specially that the data set is for a big company that has a huge number of sub-companies in almost all the middle east countries and these extreme values makes sense and will not be applicable, so we've decided to keep them instead of discarding them.

Part 4: Business Questions & Visualization

`Now, It's time to figure out answers for the company busniess questions.`

- Ask nine interesting questions about the dataset.
- Answer the questions by using Numpy/Pandas to compute the answers or Matplotlib/Seaborn to plot graphs.
- Create new columns, combine multiple datasets, and perform grouping/aggregation when needed.

Question 1: Who are the top five most popular suppliers?

Procedure: Trying to figure out which suppliers are the most popular means determining which suppliers the company buys the most from. The extended cost is the most important factor to consider.

This provider is regarded as a trusted supplier since he delivers items or services on time, despite the higher cost. As a result, he will be the most popular supervisor. Because there are so many different suppliers who have purchased from them, the following processes will be used to evaluate the top five suppliers, according to the dataset:

Using the functions **gorupby(), sum(), and sort values()**

- sort the sum value in ascending order by grouping supplier number column and calculating the sum of the extended cost for each supplier number.'

```
[91]: top_suppliers = (
          df.groupby("supplier_no", as_index=True)["extended_cost"]
          .sum()
          .sort_values(ascending=False)
      )
```

```
[92]: pd.DataFrame(top_suppliers[:5]).round()
```

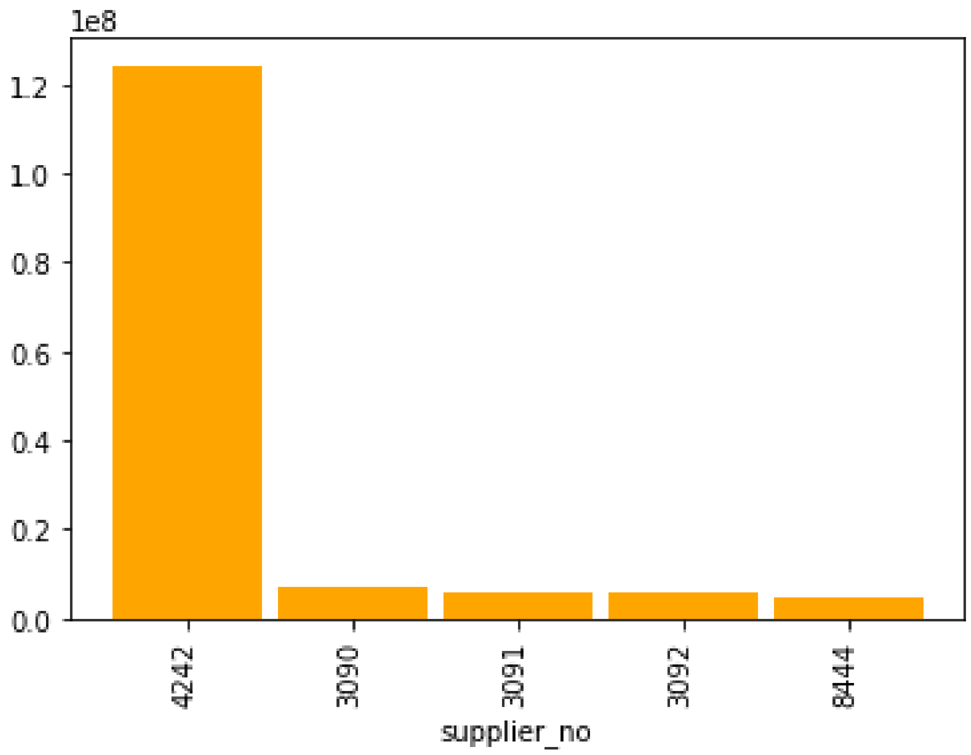
```
[92]:              extended_cost
      supplier_no
      4242           124406980.0
      3090             6930100.0
      3091             6050000.0
      3092             5720000.0
      8444             4540876.0
```

Answer:

As can be seen, the company purchased the most from supplier number 4242 , followed by 3090 .

```
[93]: top_suppliers.head().plot.bar(color="orange", width=0.9)
```

```
[93]: <AxesSubplot:xlabel='supplier_no'>
```



The plot bar figure illustrating the above result demonstrates that suppliers Nos. (4242, 3090, 3091, 3092, and 8444) are the most popular suppliers.

Question Two: Which are the top five companies in terms of purchases?

Procedures:

- Splitting and grouping by company name column, then using the sum function to calculate the sum of each business's extended cost to determine which company has the highest extended cost.
- Store the result in an object.
- Sort the objects in the returned list in ascending order.
- From the specified series object, create a dataframe and print the first three companies.

```
[94]: total_purchased = df.groupby("company_name")["extended_cost"].sum()
```

```
[95]: print("\n TOP 5 Purchaced Companies")

total_purchased.sort_values(ascending=False).to_frame().head(5).round()
```

```

 TOP 5 Purchaced Companies
```

```
[95]:              extended_cost
company_name
Jordan           128292194.0
Saudi Arabia      26084595.0
Qatar               500100.0
UAE                 395502.0
Kuwait              332269.0
```

Answer:

Jordan, Saudi Arabia, Qatar, the United Arab Emirates, and Kuwait had the most orders purchased, as displayed..

Convert objects to integer data types using the seaborn library, and then use the index to access all of the values in the converted object to set it in the plot's x-axis. In the y-axis, save the extended cost for each company.

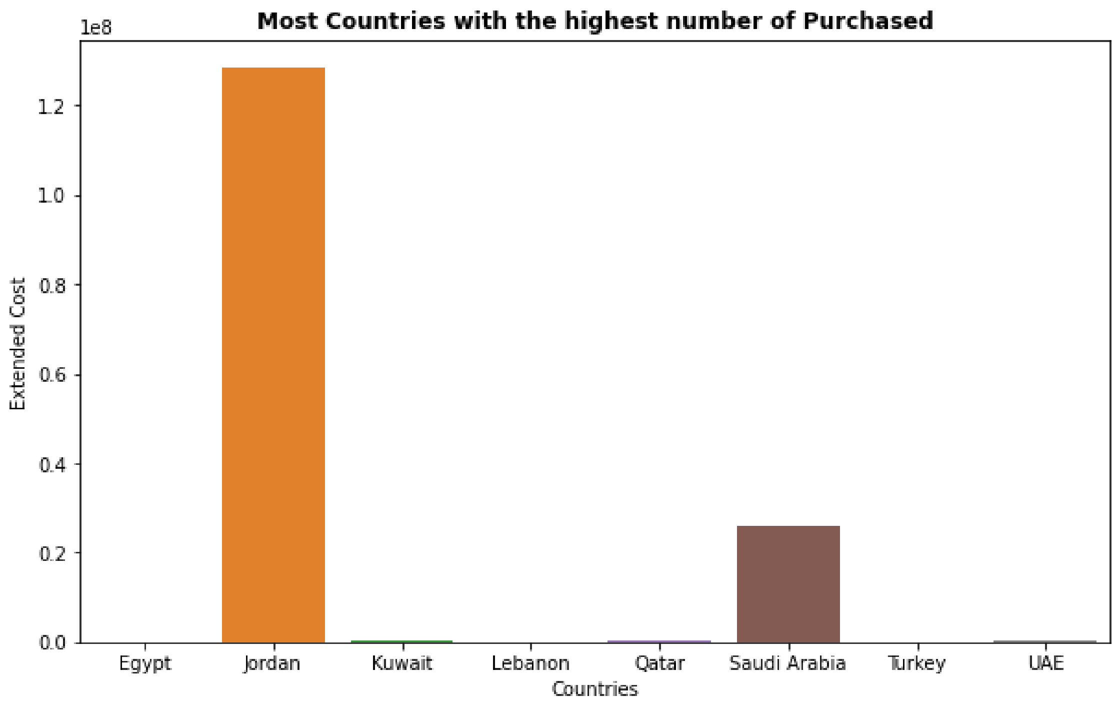
```
[96]: plt.figure(figsize=(10, 6))

sns.barplot(x=total_purchased.index, y=total_purchased.values, data=df)

plt.title("Most Countries with the highest number of Purchased", weight="bold")

plt.xlabel("Countries")
plt.ylabel("Extended Cost")
```

```
[96]: Text(0, 0.5, 'Extended Cost')
```



Question Three: Most three active years for purchase, and how much was costed that year??

Procedure:

The general process involves the following steps:

- Apply the function '.sum()' of extended cost for each year to group and separate the year of order data using the pandas 'groupby' method.
- Combine the results into a data frame.

[97]:
```
best_year = pd.DataFrame(
    df.groupby("year_of_order", as_index=True)["extended_cost"].sum()
)
```

[98]:
```
plotet_result = best_year.sort_values(by="extended_cost", ascending=False)
plotet_result[:3].round()
```

[98]:
```
               extended_cost
year_of_order
2022             126817440.0
2016              22422906.0
2018               2270000.0
```

Plot the data and use the 'kind ='pie" keyword to make a pie chart, as well as the following additional parameters:

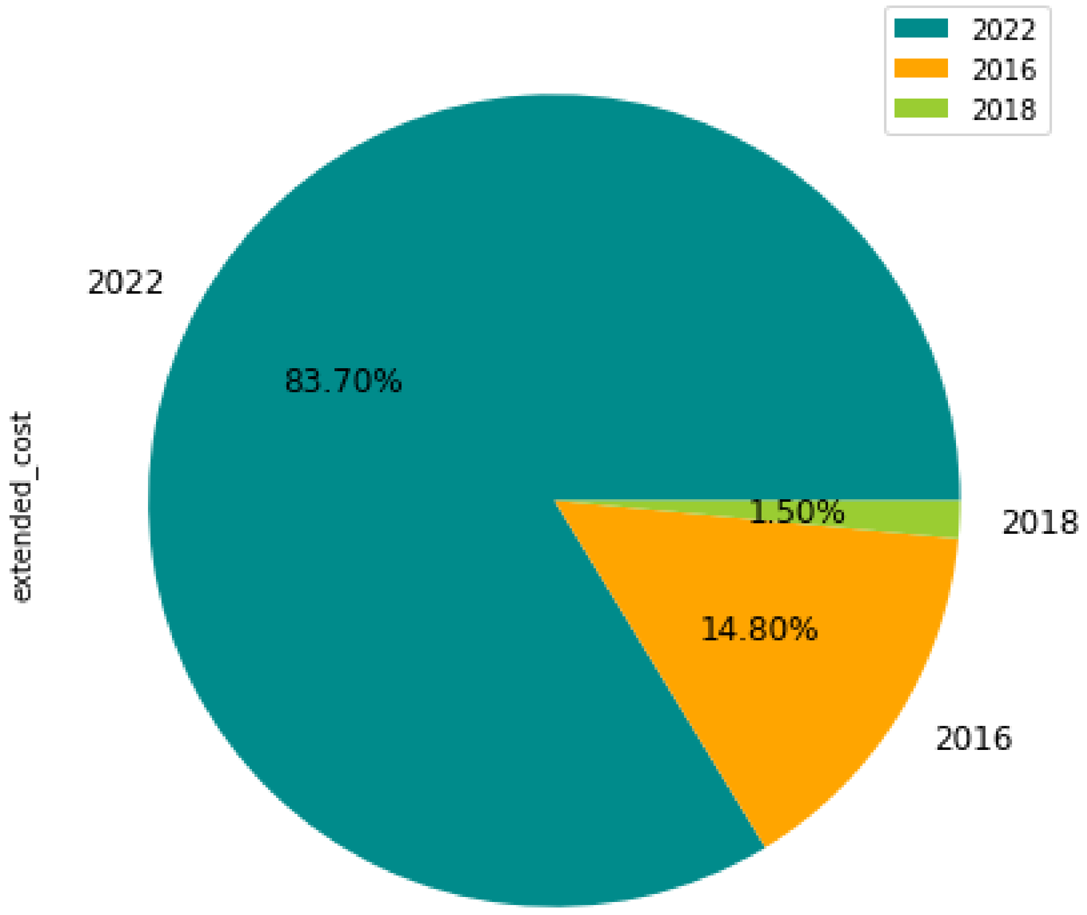
1. `autopct`: For each pie wedge, use string formatting to represent the percent number for the data.

2. `figsize`: fontsize, colors: Adjust the figure size, font size, and color scheme of the chart.

3. `subplots`: To draw the pie plots for each column as subplots.

```
[99]: plotet_result[:3].round().plot(
          kind="pie",
          figsize=(10, 7),
          colors=["darkcyan", "orange", "yellowgreen"],
          fontsize=12,
          autopct="%.2f%%",
          subplots=True,
      )
```

```
[99]: array([<AxesSubplot:ylabel='extended_cost'>], dtype=object)
```



Answer:

The biggest amounts purchased were in the year 2022, with an 83 percent percentage of costs, and secondly in 2016, with a 14.80 percent percentage of costs, as shown by the statistics above.

Question Four: How many orders have been canceled?

Procedures:

The general process involves the following steps:

- Using the lambda function and the order no column, which holds the C letter for canceled products, to store canceled orders in a data frame column.
- In an object, keep track of the number of data frame rows.
- To find the number of canceled orders, divide the total number of data frame rows by the number of canceled orders.

[100]:
```
df["order_canceled"] = df["order_no"].apply(lambda x: int("C" in x))
```

[101]:
```
c_orders = df["order_canceled"].sum()

rows_no = df.shape[0]

canclled_perc = c_orders / rows_no * 100
```

[102]:
```
print(
    "Number of orders canceled: {}/{} ({:.2f}%) ".format(
        c_orders, rows_no, canclled_perc
    )
)
```

```
Number of orders canceled: 8/742 (1.08%)
```

Plot the result using `pie chart` by passing the following:

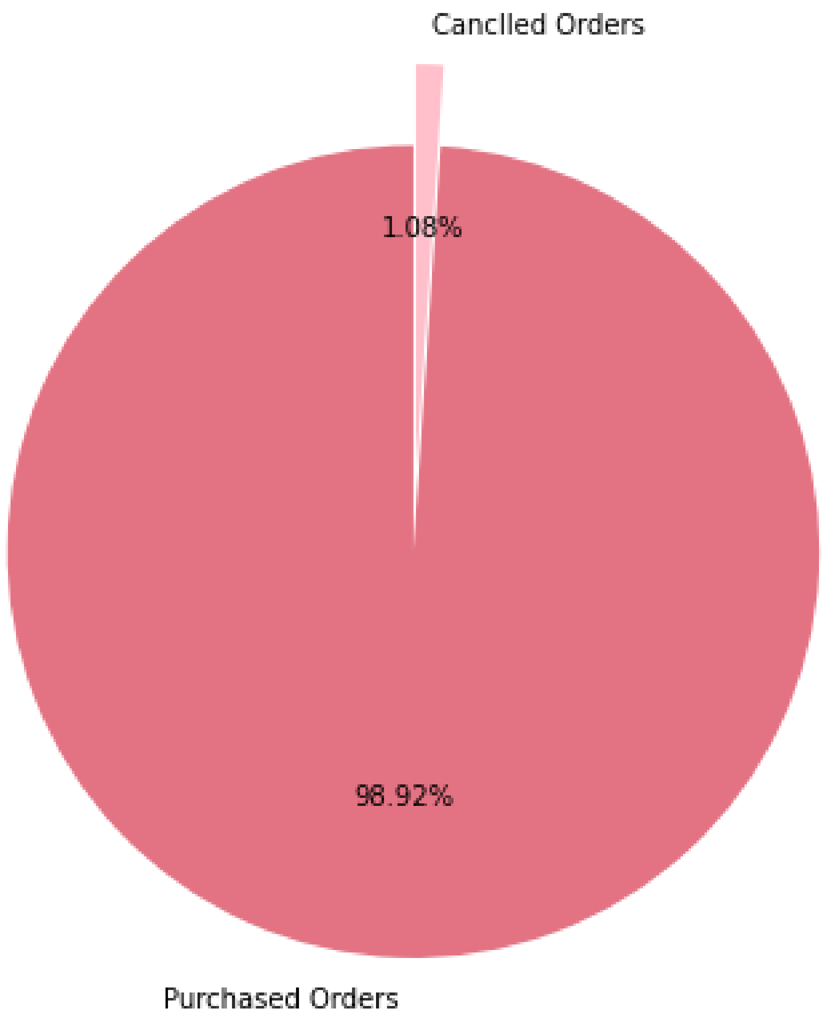
- ordered_cancled column values.
- labels to be presented in the chart.
- explode: is a tuple in which each element corresponds to a pie chart wedge.
- The colors of the pie chart.
- The angle.
- autopct: To display the percentage value.

[103]:
```
plt.figure(figsize=(14, 7))

labels = ["Purchased Orders", "Canclled Orders"]

plt.pie(
    df["order_canceled"].value_counts(),
    labels=labels,
    explode=[0.1, 0.1],
    autopct="%1.2f%%",
    colors=["#E37383", "#FFC0CB"],
    startangle=90,
)

plt.show()
```



Answer:

With only eight canceled orders, we have a ’low percentage of canceled orders (1.08 percent). Examining these canceled purchases could help avoid cancellations in the future.

Question Five: Which most five preferred purchaced orders

Procedures:

The general process involves the following steps:

- Split and group the data in the 'description' column for the purchased products.
- Make a descending order of the findings.
- Show indices form 0 to 4
- Plot the result

```
[104]: purchased_or = pd.DataFrame(df.groupby("description").sum()["quantity_ordered"])
```

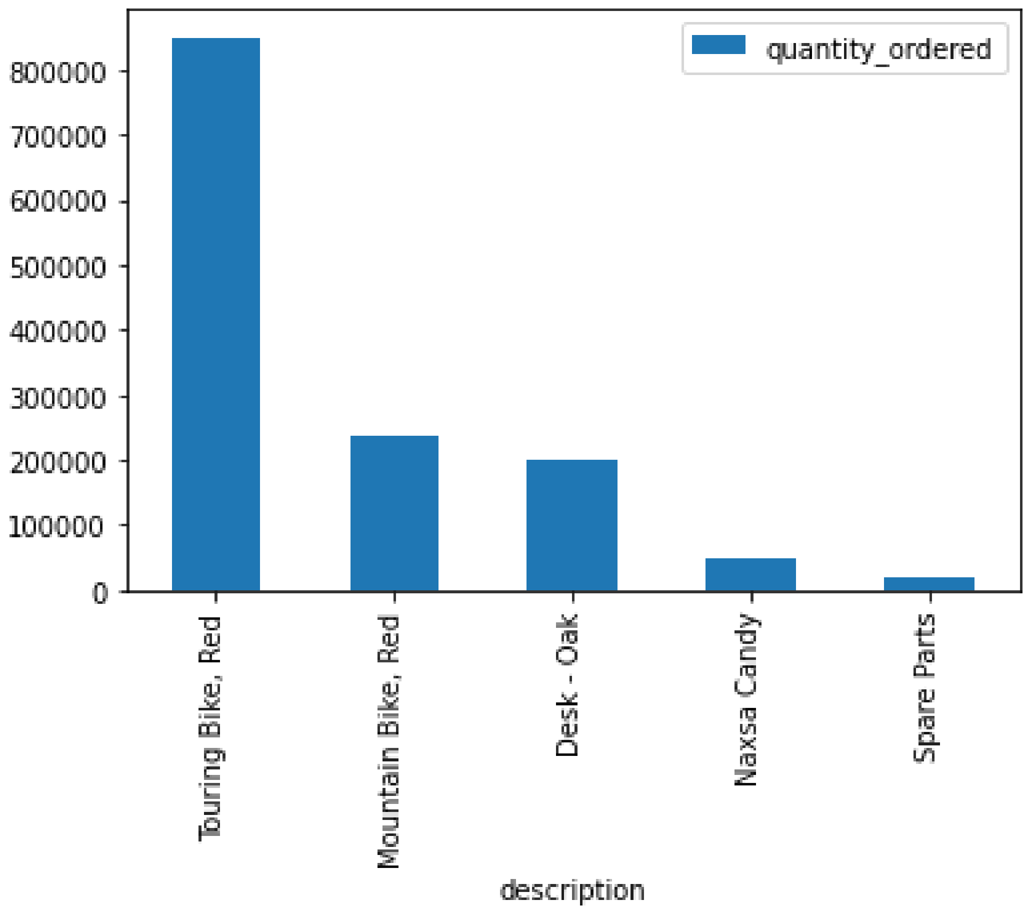
```
[105]: purchased_or = purchased_or.sort_values("quantity_ordered", ascending=False)
```

```
[106]: purchased_or[:5].round()
```

```
[106]:                     quantity_ordered
       description
       Touring Bike, Red           850201.0
       Mountain Bike, Red          236424.0
       Desk - Oak                  200000.0
       Naxsa Candy                  48300.0
       Spare Parts                  21000.0
```

```
[107]: purchased_or[:5].round().plot.bar()
```

```
[107]: <AxesSubplot:xlabel='description'>
```



Answer:

Red Touring bike, Red mountain bike, Desk, Naxsa candy and spare parts were the top five preferred purchase orders from the companies.

Question Six: What is the overall purchasing trend?

[108]:
```
print("Maximum order amount :", df["quantity_ordered"].max())
print("Minumum order amount :", df["quantity_ordered"].min())
print("Average order value :", df["quantity_ordered"].mean())
print("Median order value :", df["quantity_ordered"].median())
```

```
Maximum order amount : 234567.0
Minumum order amount : -3.47
Average order value : 1928.4532479784364
Median order value : 100.0
```

Answer:

From the preceding result, we can make a few significant points as mentioned.

Question Seven: Which most five preferred OP orders?

Procedures:

The general process involves the following steps:

- Apply the result to a one-dimensional array holding all the item numbers for this order type. * Group order type column, which has different types of all purchased things based on the item number.
- Store item numbers for order type (OP), which is only the purchase order, because other order types (OR: Order Requisition), (OB: Blanket Order), and (OQ: Order Quotation) must be eliminated from our analytical procedure.
- Make an empty dictionary so we can have a unique reference to other values.
- Convert the OP_order array to a NumPy array.
- Create two objects: one to maintain track of the NumPy array's unique values, and the other to keep track of the count.

Note: Use 'np.unique' to find the unique items of the OP orders array and return a sorted array of these elements, as well as a second array containing the counter for each element in the first array.

To assign the two arrays to the empty dictionary as two sequences, use the dict () and zip () methods. This manner, we'll be able to create a key with the item number associated with the OP order, as well as a value that represents the count for that item number.

- Sort the key-value pairs in the dictionary into tuples in a list and display them in descending order.
- Because the dictionary's goal is to sort it by value, we'll utilize the built-in item getter method to sort the newly generated dictionary by value.
- Save the first five elements of the keys as well as the first five elements of the values.

[109]:
```
or_type = df.groupby("order_type")["item_no"].apply(list)
```

[110]:
```
OP_orders = or_type["OP"]
count_op = {}
```

[111]:
```
OP_orders = np.array(OP_orders[:])

unique, counts = np.unique(OP_orders, return_counts=True)

count_op = dict(zip(unique, counts))
```

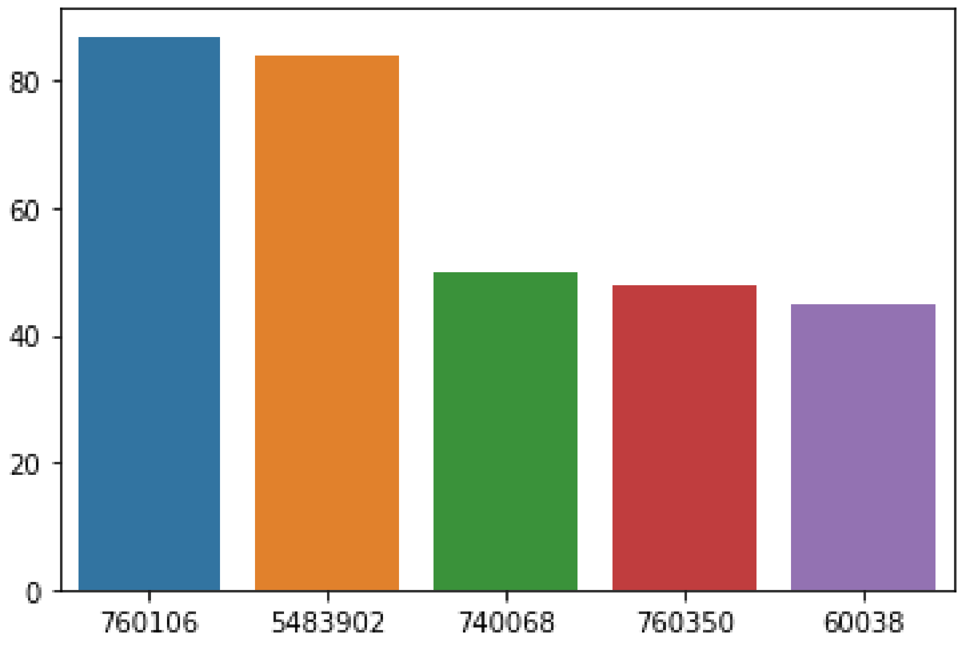
[112]:
```
sorted_count_view = sorted(count_op.items(), key=operator.itemgetter(1),␣
 ↪reverse=True)
```

[113]:
```
x = [i[0] for i in sorted_count_view[:5]]
```

```
y = [i[1] for i in sorted_count_view[:5]]
```

```
[114]: sns.barplot(x, y, order=x)
```

```
[114]: <AxesSubplot:>
```



Answer:

OP orders were the most common for item number 760106, followed by the other item numbers as shown.

Question Eight: How many times has the company used a purchase requisition Form to buy goods and services?

Procedures:

The steps in the general procedure are as follows:

- Because the purchase requisition form is so significant in the procurement process, it's crucial to determine how many times the organization has created an OR order type (Requisition Order) in conjunction with the purchased orders.
- In the dataframe, construct a column labeled 1 for the counts of OR order types, and a column labeled 0 for the numbers of other order types.

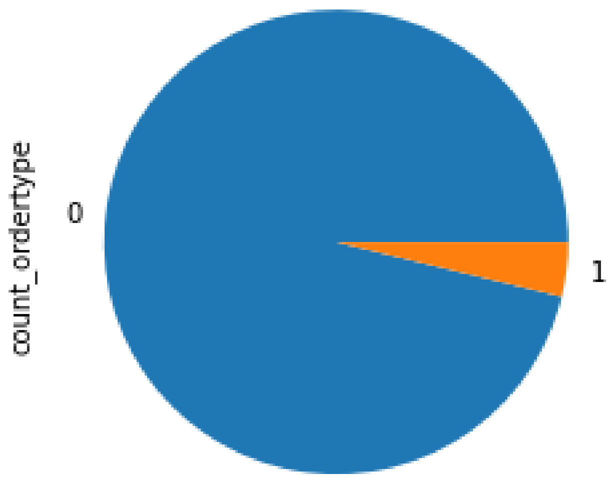
```
[115]: df["count_ordertype"] = np.where(df.order_type == "OR", 1, 0)
       df.count_ordertype.value_counts()
```

[115]:
```
0    714
1     28
Name: count_ordertype, dtype: int64
```

The majority of the orders have been placed with OR, indicating that the company has a good purchasing process.

[116]:
```
df.count_ordertype.value_counts().plot(kind="pie")
```

[116]:
```
<AxesSubplot:ylabel='count_ordertype'>
```



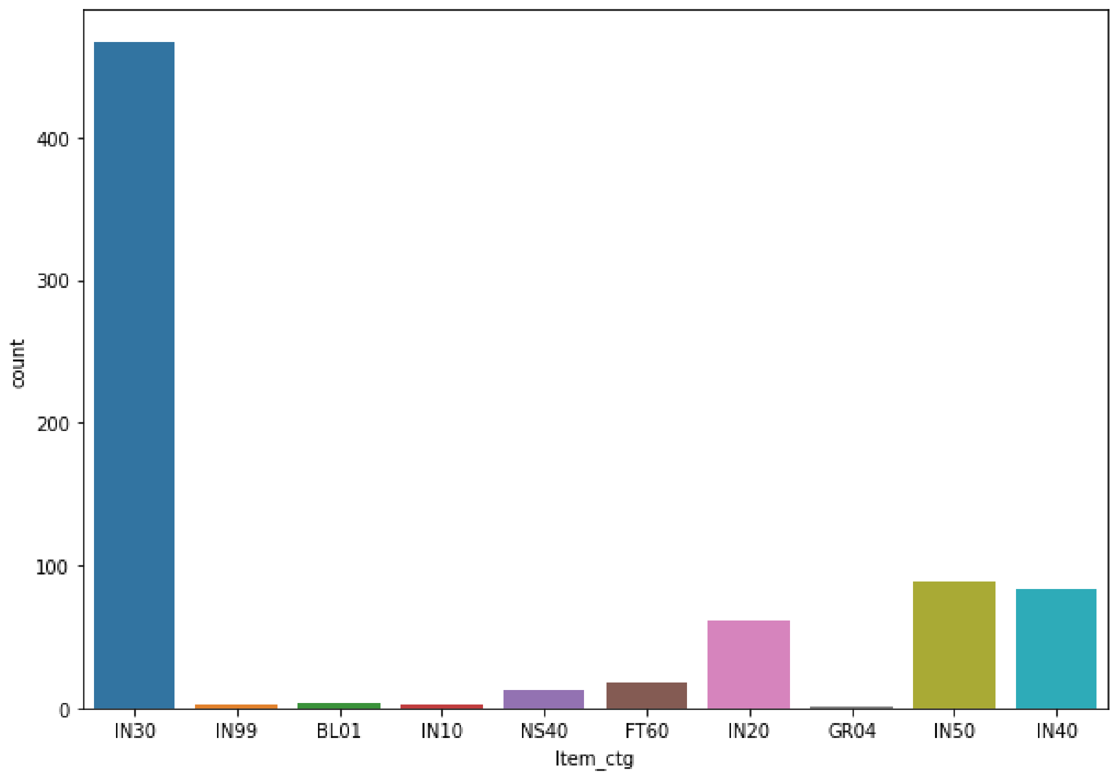
Question Nine: What are the most popular item categories purchased by companies?

Procedures:

- Using the seaborn library, create a bar chart using the counts of observations in the Item category column using the countplot() method.

[155]:
```
plt.figure(figsize=(10, 7))
sns.countplot(df["Item_ctg"])
```

[155]:
```
<AxesSubplot:xlabel='Item_ctg', ylabel='count'>
```



Answer:

The item category IN30 is definitely the most purchased, followed by IN50 and IN40, as shown in the second graph.

Conclusion

The objective of the EDA process has been reached, and main questions have been answered so this process was useful and important to the headoffice company and it's branches. The successful implementation of the EDA purchase order process benefited the organization in a variety of ways. Financial data quality has improved and is now more accurate than before as well as imporved the purchase order process.

Once the organization has the insights and analysis now, they have a big image on the details, they are ready to begin reducing the number of suppliers by selecting the preferred ones as well as when order quantities per preferred supplier increase, it presents opportunities for negotiating for better cost discounts, payment processing terms, and other terms with a preferred supplier.

In order to continue improving the PO process, I have proposed the following addi-

tional recommendations:

Regarding to the companies that have high purcahse orders with a high extended cost, company must check their inventory level and try to reduce their inventory level, in order to effectively purchase the right quantity of goods and services for coming purchase process, as storing products without using them consumes a company's resources, adds more costs like renting cost and reduce its cash flow.

It's better for the company to reduce their purchasing costs and time by selecting fewer suppliers and excluding those who cancelled orders and did not comply with orders and increasing the level of cooperation in supplier relationships and using an integrated database. As a result, lowering the purchase price contributes to lowering the final price of products.

```
[ ]:
```