Note: Answers to question 1 (a-c) are all at the bottom of this notebook

In [1]:

```
import pandas as pd
import numpy as np
import seaborn as sns
```

Read in the data and check it's format for useability

- check the variable types
- check for any missing data

In [2]:

```
shoe_orders_df = pd.read_csv('q1_data_set.csv')
shoe_orders_df.head()
```

Out[2]:

	order_id	shop_id	user_id	order_amount	total_items	payment_method	created_at
0	1	53	746	224	2	cash	2017-03-13 12:36:56
1	2	92	925	90	1	cash	2017-03-03 17:38:52
2	3	44	861	144	1	cash	2017-03-14 4:23:56
3	4	18	935	156	1	credit_card	2017-03-26 12:43:37
4	5	18	883	156	1	credit_card	2017-03-01 4:35:11

In [3]:

```
shoe_orders_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 7 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   order_id        5000 non-null   int64
 1   shop_id         5000 non-null   int64
 2   user_id         5000 non-null   int64
 3   order_amount    5000 non-null   int64
```

```
 4   total_items     5000 non-null   int64
 5   payment_method  5000 non-null   object
 6   created_at      5000 non-null   object
dtypes: int64(5), object(2)
memory usage: 273.6+ KB
```

In [4]:

```
print('created_at dtype:', type(shoe_orders_df['created_at'][0]))
print('Different payment methods:', shoe_orders_df['payment_method'].unique
```

```
created_at dtype: <class 'str'>
Different payment methods: ['cash' 'credit_card' 'debit']
```

- Order amount and items are numerical - no transformations needed here
- The date field is a string and should be converted for better use
- Payment methods can stay as string values, they are not needed as integers (no label encoding needed)

Clean the data

- Before looking into anything, the data should be cleaned
- Extract information from the created_at to have individual numerical features (easy to use later on)

In [5]:

```
from typing import List
def extract_time(df: pd.DataFrame, col_name: str = 'created_at', drop: bool
    """ Extract date/time data from the created_at column

    Args:
        df: The current dataframe in use
        col_name: The column to extract time data from
        drop: If true, drop the original time column

    Returns:
        A new dataframe containing new column with separate information abo
    """
    df = df.copy()

    datetime_col = pd.to_datetime(df[col_name])
    # extract date info
    df['day'] = datetime_col.dt.day
    df['month'] = datetime_col.dt.month
    df['year'] = datetime_col.dt.year
    df['weekday'] = datetime_col.dt.dayofweek
    df['year_day'] = datetime_col.dt.dayofyear

    # extract time info; ignore the seconds (different in seconds sholud no
    df['hours_time'] = datetime_col.dt.hour + (datetime_col.dt.minute / 60)

    if drop:
```

```
        df.drop(columns=[col_name], inplace=True)

    return df
orders_df = extract_time(shoe_orders_df)
orders_df.head()
```

Out[5]:

	order_id	shop_id	user_id	order_amount	total_items	payment_method	day	month
0	1	53	746	224	2	cash	13	3
1	2	92	925	90	1	cash	3	3
2	3	44	861	144	1	cash	14	3
3	4	18	935	156	1	credit_card	26	3
4	5	18	883	156	1	credit_card	1	3

In [6]:

```
orders_df.shape
```

Out[6]: (5000, 12)

EDA

- Perform some exploratory data analysis to get insight into the data distribution

In [7]:

```
import seaborn as sns
sns.set_style('darkgrid')
# graph each order against order_amount to see how order order amount is di
price_by_order = sns.scatterplot(x='order_id', y='order_amount', data=order
price_by_order.set_title('Order vs Price')
price_by_order.set_ylabel('Order Price')
```

Out[7]: Text(0, 0.5, 'Order Price')

- From this first scatter plot, outliers are evident
- There are a few orders at $700,000, and quite a few that seem to be between $20,000 - $100,000

Group data by specific columns

- Since there are many unique orders, examine the order amount by shop id, user id and total items
- When calculating the mean in each of these 3 columns, factors affecting the AOV will start to become apparent

In [8]:

```
avg_cost_by_shop = orders_df.groupby('shop_id')['order_amount'].mean().sort
avg_cost_by_user = orders_df.groupby('user_id')['order_amount'].mean().sort
avg_cost_by_quantity = orders_df.groupby('total_items')['order_amount'].mea
```

Shop Aggregation

In [9]:

```
avg_cost_by_shop.head()
```

Out[9]:

```
shop_id
42     235101.490196
78      49213.043478
50        403.545455
```

```
90       403.224490
38       390.857143
Name: order_amount, dtype: float64
```

Shop Aggregation: Findings

- 2 shops have much higher mean order costs than the rest

User Aggregation

In [10]:

```
avg_cost_by_user.head()
```

Out[10]:

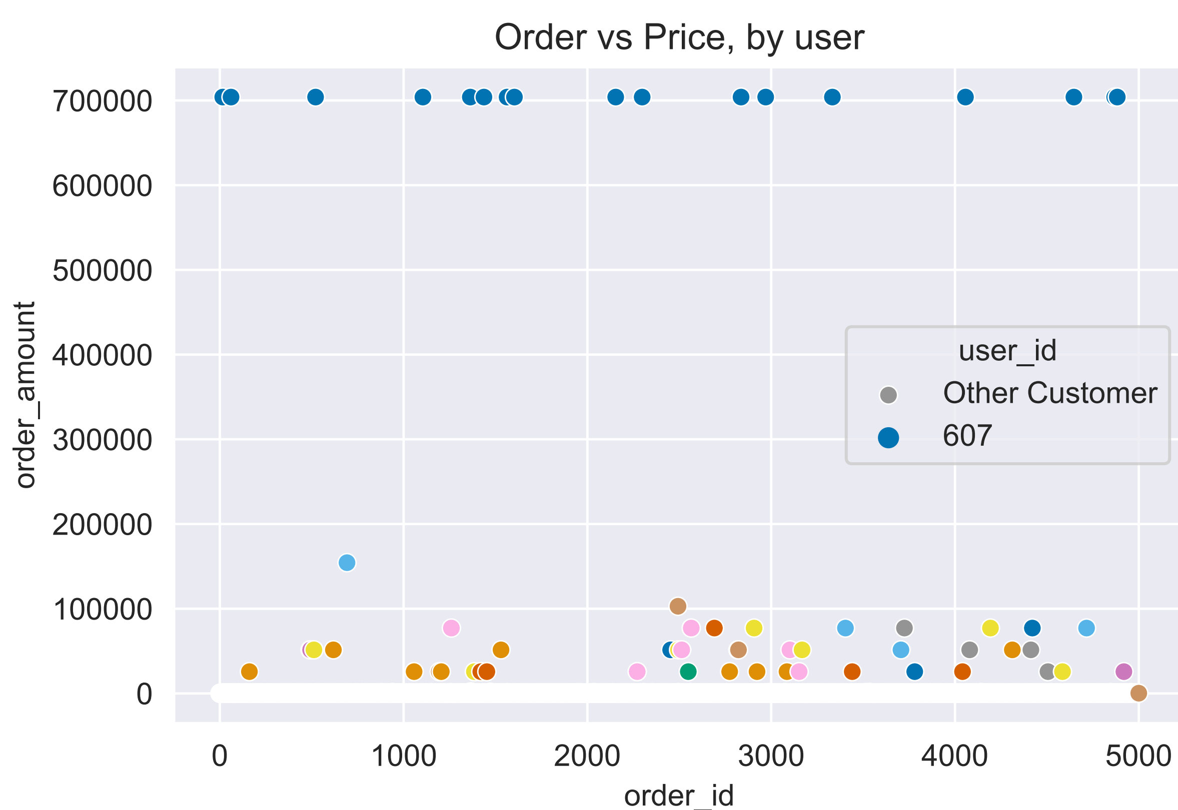
```
user_id
607    704000.000000
878     14266.909091
766      8007.600000
834      6019.000000
915      5785.142857
Name: order_amount, dtype: float64
```

In [11]:

```
# same graph as the first one except with user_id as the hue
price_by_order = sns.scatterplot(x='order_id', y='order_amount', data=order
price_by_order.set_title('Order vs Price, by user')
price_by_order.legend(['Other Customer', '607'], title='user_id')
```

Out[11]:

```
<matplotlib.legend.Legend at 0x20a0ebd7bc8>
```



User Aggregation: Findings

- One specific user (607) has a mean order value of $704,000 which is much higher than the rest

Quantity Aggregation

In [12]:

```
avg_cost_by_quantity.head()
```

Out[12]:

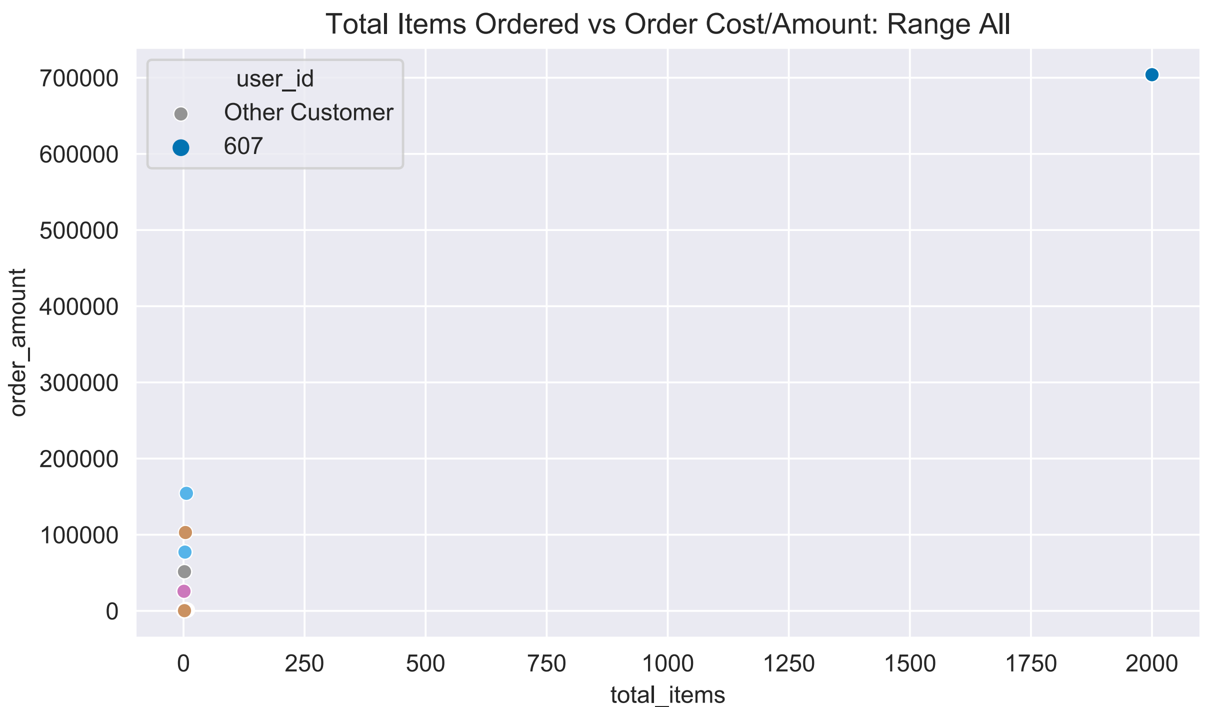
```
total_items
2000    704000.000000
6        17940.000000
3         1191.076514
8         1064.000000
4          947.686007
Name: order_amount, dtype: float64
```

In [13]:

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots(2, figsize=(8, 10))

quantity_v_amount_ax = sns.scatterplot(x='total_items', y='order_amount', h
quantity_v_amount_ax.set_title('Total Items Ordered vs Order Cost/Amount: R
quantity_v_amount_ax.set_xlim(0, 10)
quantity_v_amount_ax.legend(['Other Customer', '607'], title='user_id')

quantity_v_amount_ax_2 = sns.scatterplot(x='total_items', y='order_amount',
quantity_v_amount_ax_2.set_title('Total Items Ordered vs Order Cost/Amount:
quantity_v_amount_ax_2.legend(['Other Customer', '607'], title='user_id')
```

Out[13]:

```
<matplotlib.legend.Legend at 0x20a0f2afd88>
```



Quantity Aggregation: Findings

- This same customer (607) that had a mean order value of $704,000 also ordered 2000 shoes on average
- Other than the one customer who has 2000 items per order, all other orders fall in the 1-8 item range per order
- The previous analysis has not revelaed any insight into price per shoe (at a certain store), so I will look into this next

Explore the effects of store cost per shoe

- Since each store sells one shoe, look at costs by shop for one shoe
- Create a new feature called price per shoe and examine it's interaction with order cost

In [14]:
```
orders_df['price_per_shoe'] = orders_df['order_amount'] / orders_df['total_

per_shoe_ax = sns.scatterplot(x='price_per_shoe', y='order_amount', hue='sh
per_shoe_ax.set_title('Price per Shoe vs Order Cost/Amount')
per_shoe_ax.legend([])
```

Out[14]: <matplotlib.legend.Legend at 0x20a0f47cb88>

Explore the effects of store cost per shoe: Findings

- One shop sells very expensive shoes, at a cost of over $25,000 per shoe while all of the other shops have a selling point for shoes that is far lower

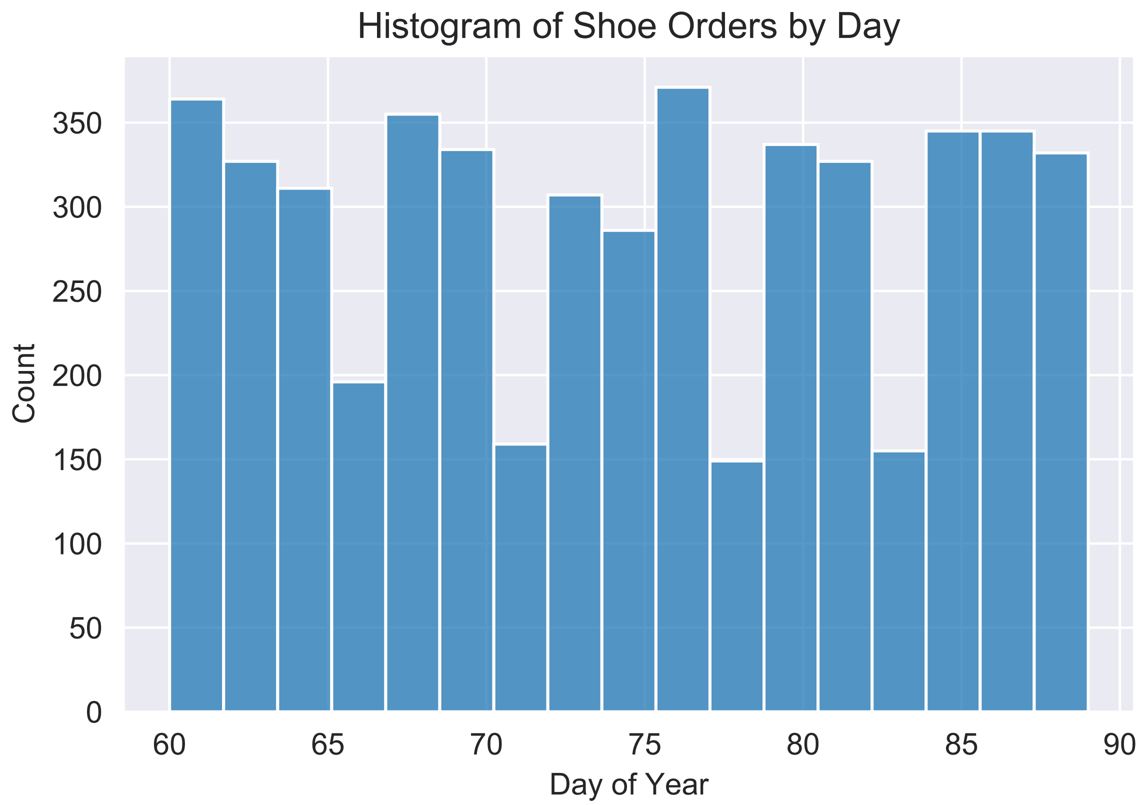
Examine any time data

- Since all the orders are made over a 30 day period, see if there is anything strange with amount of orders on certain days or time of day

In [15]:
```
# we only have data made from 2017, see the amount of orders by month
orders_by_day = sns.histplot(x='year_day', data=orders_df)
```

```
orders_by_day.set_title('Histogram of Shoe Orders by Day')
orders_by_day.set_xlabel('Day of Year')
```

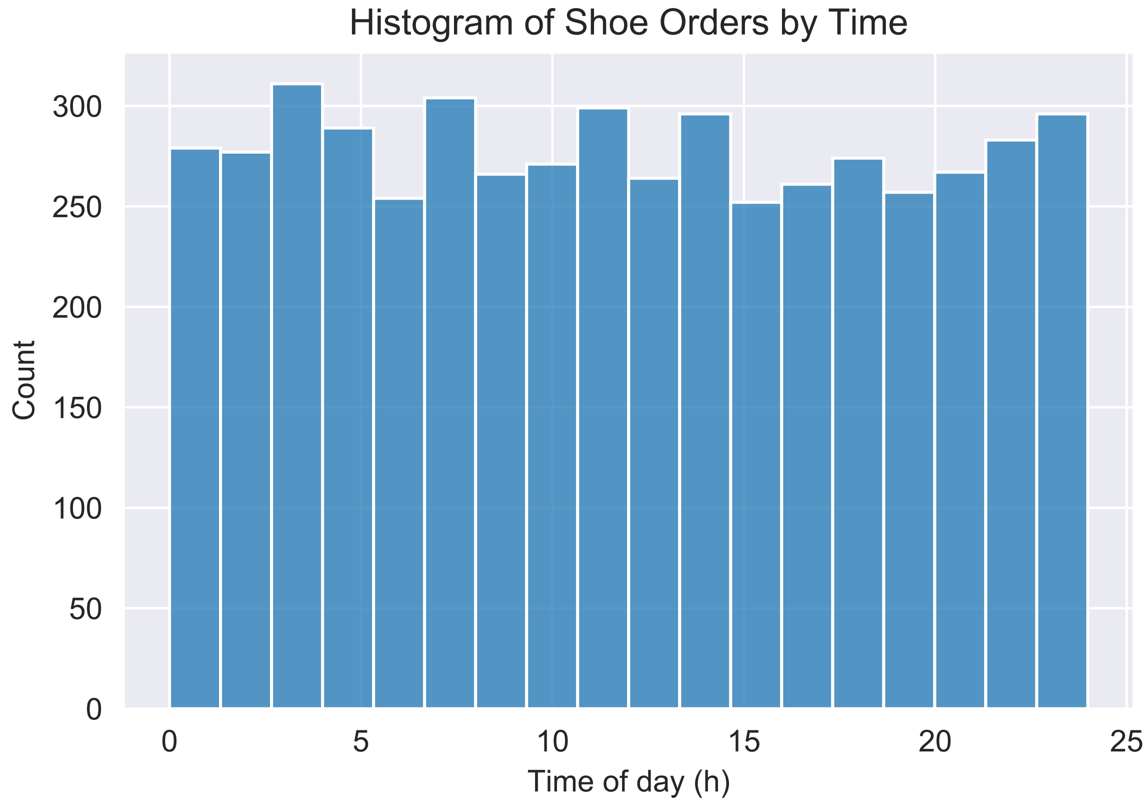
Out[15]: Text(0.5, 0, 'Day of Year')



In [16]:

```
orders_by_hour = sns.histplot(x='hours_time', data=orders_df)
orders_by_hour.set_xlabel('Time of day (h)')
orders_by_hour.set_title('Histogram of Shoe Orders by Time')
```

Out[16]: Text(0.5, 1.0, 'Histogram of Shoe Orders by Time')



Examine any time data: Findings

- After drawing histograms for days and times of orders, there seems to be nothing for date or time that points to outliers in the data
- The final thing to examine would be payment methods since date/time do not seem relevant

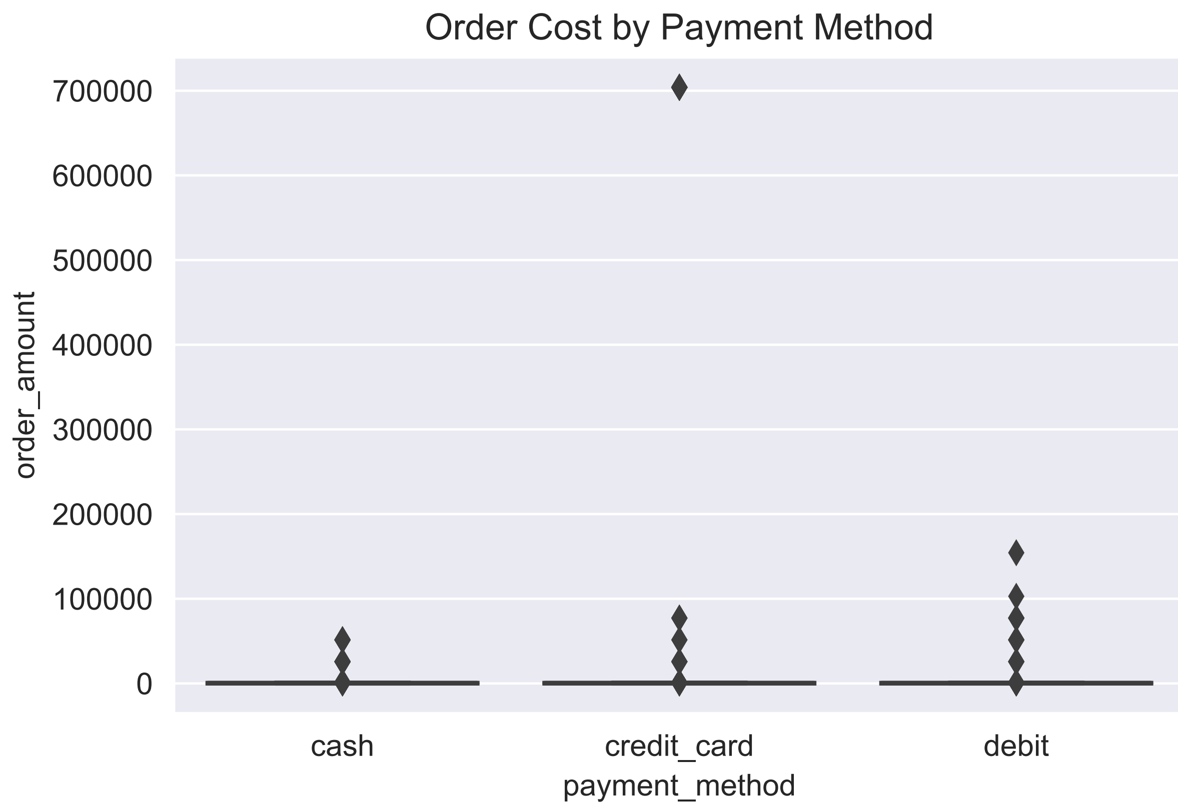
Examine payment methods

- See if any of the payment methods had differences

In [17]:

```
orders_by_payment = sns.boxplot(x='payment_method', y='order_amount', data=
orders_by_payment.set_title('Order Cost by Payment Method')
```

Out[17]: `Text(0.5, 1.0, 'Order Cost by Payment Method')`



Examine payment methods: Findings

- There is one huge outlier with credit card payments but the order amount tells us that this is the same user (607) who made a very large number of shoe orders
- In general, the payment methods do not reveal any new insights into the data

Look at the standard deviation of the order amount to see how spread the data is

- To quantify how much outliers are affecting the average order value (AOV) calculate the standard deviation
- Note: High standard deviations can be highly affected by outliers

In [18]:

```
order_cost_std = orders_df['order_amount'].std()
order_cost_mean = orders_df['order_amount'].mean()
order_quantity_std = orders_df['total_items'].std()
order_quantity_mean = orders_df['total_items'].mean()
print('Order cost standard dev: $' + str(round(order_cost_std, 2)))
print('Order cost mean: $' + str(round(order_cost_mean, 2)))
print('Order quantity standard dev: ' + str(round(order_quantity_std, 2)) +
print('Order quantity mean: ' + str(round(order_quantity_mean, 2)) + ' item
```

```
Order cost standard dev: $41282.54
Order cost mean: $3145.13
```

```
Order quantity standard dev: 116.32 items
Order quantity mean: 8.79 items
```

- Both the order cost mean and order quantity mean are very high
- The standard deviation for both order cost and order quantity is also very high. From the previous data exploration, outliers are the cause of this
- Next, examine the effects of removing the outliers on standard deviation

To get the outliers, I can train an isolation forest on the order amount

- Certain shops and users have higher average order values
- From the first scatter plot it seems like there are around 1% of samples that are outliers, train an isolation forest to identify these outliers

In [19]:

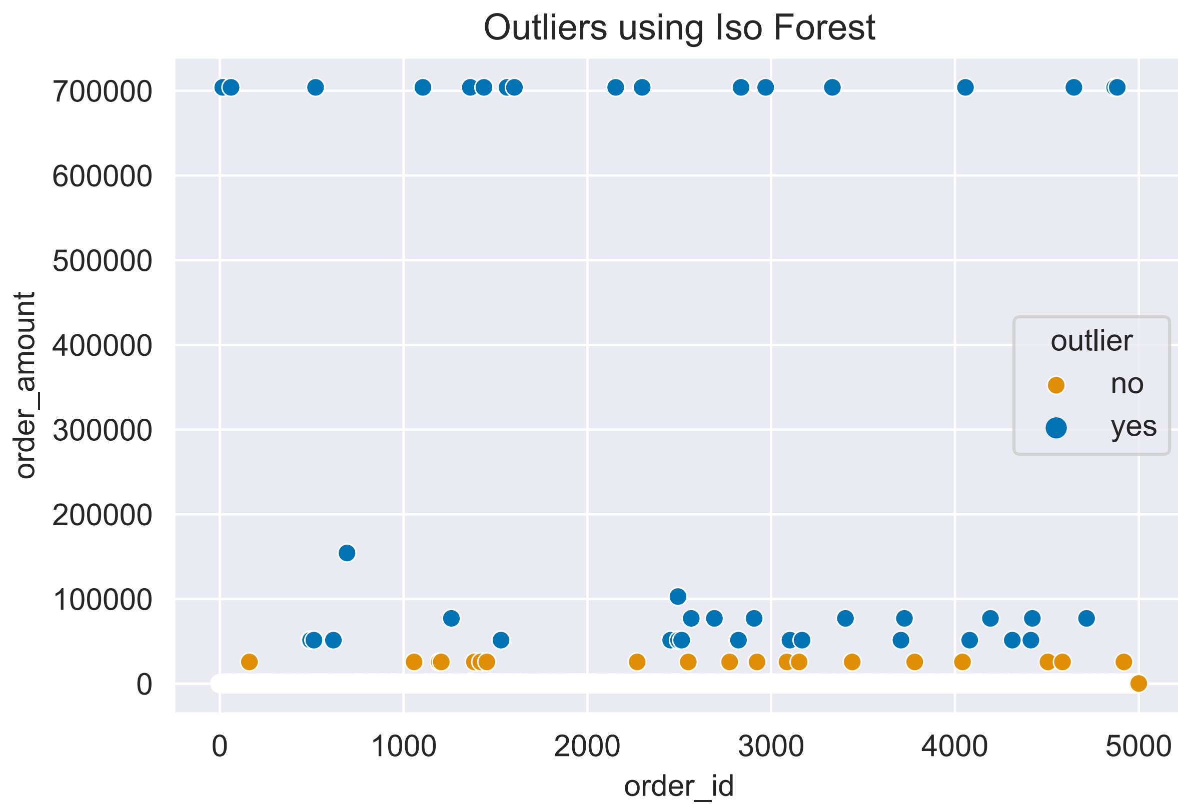
```
from sklearn.ensemble import IsolationForest

def make_isolation_forest(df: pd.DataFrame, n_estimators: int, contaminatio
    # create an isolation forest model
    detector = IsolationForest(n_estimators=n_estimators, contamination=con
    detector.fit(df[['order_amount']])
    # use the fitted isolation forest and add it to the df to see the outli
    outlier_df = df.copy()
    outlier_df['outlier'] = detector.predict(outlier_df[['order_amount']])
    outlier_df['score'] = detector.decision_function(outlier_df[['order_amo

    return outlier_df

# add the new columns from the isolation forest to a df - isolation forest
outlier_df = make_isolation_forest(orders_df, 50, 0.01)
```

In [20]:

```
outlier_ax = sns.scatterplot(x='order_id', y='order_amount', hue='outlier',
outlier_ax.set_title('Outliers using Iso Forest')
outlier_ax.legend(['no', 'yes'], title='outlier')
```

Out[20]:

```
<matplotlib.legend.Legend at 0x20a11722388>
```



Isolation Forest Scatter Plot: Findings

- After training an isolation forest with 1% contamination, it is evident that the majority of the outliers have been detected
- To examine the outliers, I can locate the outliers in the data and sort by order amount, price per shoe, and order quantity

In [21]:

```
only_outlier_df = outlier_df.loc[outlier_df['outlier'] == -1]
only_outlier_df.sort_values(by='order_amount', ascending=False).head()
```

Out[21]:

	order_id	shop_id	user_id	order_amount	total_items	payment_method	day	mon
15	16	42	607	704000	2000	credit_card	7	
1562	1563	42	607	704000	2000	credit_card	19	
4868	4869	42	607	704000	2000	credit_card	22	
4646	4647	42	607	704000	2000	credit_card	2	
4056	4057	42	607	704000	2000	credit_card	28	

Outlier Order Cost/Amount Sorting: Findings (Above)

- The most expensive order amounts come from user 607 with order amounts of $704,000
- These orders were also all placed to the same shop, with shop id of 42 and all paid for with a credit card

In [22]:
```
only_outlier_df.sort_values(by='total_items', ascending=False).head()
```

Out[22]:

	order_id	shop_id	user_id	order_amount	total_items	payment_method	day	mon
15	16	42	607	704000	2000	credit_card	7	
1562	1563	42	607	704000	2000	credit_card	19	
4868	4869	42	607	704000	2000	credit_card	22	
4646	4647	42	607	704000	2000	credit_card	2	
4056	4057	42	607	704000	2000	credit_card	28	

Outlier Order Quantity/Total Items Sorting: Findings (Above)

- The largest quantity of orders come from user 607 with order quantities of 2000 shoes

In [23]:
```
only_outlier_df.sort_values(by='price_per_shoe', ascending=False).head()
```

Out[23]:

	order_id	shop_id	user_id	order_amount	total_items	payment_method	day	mon
2690	2691	78	962	77175	3	debit	22	
3403	3404	78	928	77175	3	debit	16	
3101	3102	78	855	51450	2	credit_card	21	
3705	3706	78	828	51450	2	credit_card	14	
2906	2907	78	817	77175	3	debit	16	

Outlier Price Per Shoe Sorting: Findings (Above)

- The highest price per shoe was $25,725/shoe
- The shop with shop id 78 sold these expensive shoes
- These purchaes were all made by different customers

- The order quantities (total items) were different between the orders
- The methods for payment was different

Calculate order cost standard deviation without the outliers

- See if the standard deviation has been reduced by removing 1% of the outlier data

In [24]:

```
iso_forest_std_1 = outlier_df.loc[outlier_df['outlier']==1, 'order_amount']
print('Standard deviation of order costs with 1% of outliers removed: $' +
```

```
Standard deviation of order costs with 1% of outliers removed: $1579.398646
15807
```

Increase contamination of Iso Forest Model

- By setting 1% contamination in the iso forest, the standard deviation for order prices is still high ($1500 is quite a lot for shoe orders)
- This may potentially be because I took a contamination value that was too low - raise the contamination value to 2% and re-evaluate

In [25]:

```
outlier_df_2 = make_isolation_forest(orders_df, 50, 0.02)
no_outliers_std = outlier_df_2.loc[outlier_df_2['outlier']==1, 'order_amoun
no_outliers_mean = outlier_df_2.loc[outlier_df_2['outlier']==1, 'order_amou
total_items_std = outlier_df_2.loc[outlier_df_2['outlier']==1, 'total_items
total_items_mean = outlier_df_2.loc[outlier_df_2['outlier']==1, 'total_item
print('Standard Dev. order cost With 2% outliers removed: $' + str(round(no
print('Mean order cost With 2% outliers removed: $' + str(round(no_outliers
print('Standard Dev. order quantity With 2% outliers removed: ' + str(round
print('Mean order quantity With 2% outliers removed: ' + str(round(total_it
```

```
Standard Dev. order cost With 2% outliers removed: $150.15
Mean order cost With 2% outliers removed: $297.67
Standard Dev. order quantity With 2% outliers removed: 0.95 items
Mean order quantity With 2% outliers removed: 1.97 items
```

Increase contamination of Iso Forest Model: Findings

- Before removing outliers, standard deviation for order cost was greater than $40,000 (very high)
- After removing 1% of outliers, standard deviation for order cost was around $1500 (still high)
- When indicating 2% of values as outliers, the standard deviation greatly drops off to $150, a value that looks much more reasonable for shoe order values

- After removing outliers, the total items and order costs have relatively low standard deviation which can be useful to generalize findings from this data set

1A: What is going wrong with AOV

- After visualizing the inital data, it is clear that there are outliers in the data casuing the mean and standard deviation to be very large
- Findings from isolation forest:

1. The user with **user_id 607 ordered 2000 shoes various times in the month from shop #42 costing $704,000 per order**. This greatly increased the average order value
2. A certain shop (specifically **shop_id #78) sold very expensive shoes - valued at $25,725 per shoe and had sales from various different customers**. This implies than even small order quantities from this shop would still drive the average order value up
3. By removing 2% of 'outliers', it is revealed that the standard deviation for **order amount goes from $40,000 to $150** and standard deviation for **order quantity goes from 114 items -> 0.94 items**. There is much less variance in both of these fields without the outliers which can be very useful to generalize results for this data set

- Due to finding outliers in both user buying habits and shoe costs, **means/averages should not be used to evalute any metric on the data**
- To better evalute this data, **methods that look for central values and ignore outliers (like medians)** would be much more appropriate

1B: What Metric?

- Since the initial task was to calculate average order value, Shopify likely wants to understand more about **average user spending habits**
- To understand these habits better, looking at the **price per shoe of an order** would remove the variability in the quantity of the order and focus on the price point that the average user is paying (removing outliers due to order quantity)
- Median order value per shoe will give direct insight into customer spending habits - by using a median I am in effect **removing the small amount outliers in shoe cost** and getting a value that is much more representative of the majority of user spending habits

Metric: Median Order Value Per Shoe

1C: Value of M(OV/S)?

- Use pandas to calculate the median (order value per shoe)

In [26]:

```
final_metric = (orders_df['order_amount'] / orders_df['total_items']).media
print('Median Order Value Per Shoe = $' + str(round(final_metric)) + '/shoe
```

```
Median Order Value Per Shoe = $153/shoe
```

This tells me that the 'average' customer buying from Shopify shoe stores is looking to pay $153/shoe