Problem Statement:

Our "Domain Sale" process is structured to help potential buyers purchase the domain they want immediately without the hassle of contacting the seller directly. A seller lists a domain for sale at a specific price in our marketplace. An interested buyer sees this domain for sale and decides to buy it. Extract various information, such as sales, budget, and variance. You can even compare sales and budgets with various attributes. Extract necessary information about products and customers. Make the necessary dashboard with the best you can extract from the data. Use various visualizations and features and make the best dashboard. Find key metrics and factors and show the meaningful relationships between attributes.

In [235…

```
!pip install openpyxl plotly -q
```

In [236…

```
!pip install jovian
```

```
Requirement already satisfied: jovian in /Users/nikhilreddyponnala/anaconda
3/lib/python3.11/site-packages (0.2.47)
Requirement already satisfied: requests in /Users/nikhilreddyponnala/anacon
da3/lib/python3.11/site-packages (from jovian) (2.31.0)
Requirement already satisfied: uuid in /Users/nikhilreddyponnala/anaconda3/
lib/python3.11/site-packages (from jovian) (1.30)
Requirement already satisfied: pyyaml in /Users/nikhilreddyponnala/anaconda
3/lib/python3.11/site-packages (from jovian) (6.0)
Requirement already satisfied: click in /Users/nikhilreddyponnala/anaconda
3/lib/python3.11/site-packages (from jovian) (8.0.4)
Requirement already satisfied: charset-normalizer<4,>=2 in /Users/nikhilred
dyponnala/anaconda3/lib/python3.11/site-packages (from requests->jovian)
(2.0.4)
Requirement already satisfied: idna<4,>=2.5 in /Users/nikhilreddyponnala/an
aconda3/lib/python3.11/site-packages (from requests->jovian) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in /Users/nikhilreddyponn
ala/anaconda3/lib/python3.11/site-packages (from requests->jovian) (1.26.1
6)
Requirement already satisfied: certifi>=2017.4.17 in /Users/nikhilreddyponn
ala/anaconda3/lib/python3.11/site-packages (from requests->jovian) (2024.7.
4)
```

In [237…

```python
## Importing the neccessary libraries
import jovian
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import plotly.express as px
import seaborn as sns; sns.set_theme()
import plotly.figure_factory as ff
from itertools import combinations
from collections import Counter
import datetime as dt
import warnings
warnings.filterwarnings('ignore')
```

Data wrangling

In [239…
```
### Loading the data
```

In [240…
```
## customers data
Customers_data = pd.read_excel('/Users/nikhilreddyponnala/Desktop/Budget Sa
                              'Customers',
                              dtype={'CustomerKey':str},
                              parse_dates=['BirthDate','DateFirstPurchase']
                              )
```

In [241…
```
## product data

Product_data = pd.read_excel('/Users/nikhilreddyponnala/Desktop/Budget Sales
                              'Product',
                              dtype={'ProductKey':str},
                              parse_dates=['StartDate']
                              )
```

In [242…
```
## sales data

Sales_data = pd.read_excel('/Users/nikhilreddyponnala/Desktop/Budget Sales A
                              'Sales',
                              dtype={'ProductKey':str,
                                     'CustomerKey':str,
                                     'PromotionKey':str,
                                     'SalesTerritoryKey':str},
                              parse_dates=['OrderDate', 'ShipDate']
                              )
Sales_data['DateKey'] = Sales_data['OrderDate'].astype(str)
```

In [243…
```
## territory data

Territory_data = pd.read_excel('/Users/nikhilreddyponnala/Desktop/Budget Sa
                              'Territory',
                              dtype={'SalesTerritoryKey':str}
                              )
```

In [244…
```
### Merging data

temp_data = pd.merge(Sales_data, Product_data, on='ProductKey', how='inner'
df = pd.merge(temp_data, Customers_data, on='CustomerKey', how='inner')
df = pd.merge(df, Territory_data, on='SalesTerritoryKey', how='inner')
```

In [245…
```
### Assessing data

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 58189 entries, 0 to 58188
Data columns (total 58 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   ProductKey            58189 non-null  object
 1   OrderDate             58189 non-null  datetime64[ns]
 2   ShipDate              58189 non-null  datetime64[ns]
 3   CustomerKey           58189 non-null  object
 4   PromotionKey          58189 non-null  object
 5   SalesTerritoryKey     58189 non-null  object
 6   SalesOrderNumber      58189 non-null  object
 7   SalesOrderLineNumber  58189 non-null  int64
 8   OrderQuantity         58189 non-null  int64
 9   UnitPrice             58189 non-null  float64
 10  TotalProductCost      58189 non-null  float64
 11  SalesAmount           58189 non-null  float64
 12  TaxAmt                58189 non-null  float64
 13  Unnamed: 13           0 non-null      float64
 14  Unnamed: 14           0 non-null      float64
 15  Unnamed: 15           58189 non-null  float64
 16  Unnamed: 16           58189 non-null  float64
 17  Unnamed: 17           0 non-null      float64
 18  Unnamed: 18           58189 non-null  float64
 19  Unnamed: 19           0 non-null      float64
 20  StandardCost_x        58189 non-null  float64
 21  List Price            58189 non-null  float64
 22  Unnamed: 22           0 non-null      float64
 23  diif std cost         58189 non-null  int64
 24  diff list price       58189 non-null  int64
 25  DateKey               58189 non-null  object
 26  ProductName           58189 non-null  object
 27  SubCategory           58189 non-null  object
 28  Category              58189 non-null  object
 29  StandardCost_y        58189 non-null  float64
 30  Color                 30747 non-null  object
 31  ListPrice             58189 non-null  float64
 32  DaysToManufacture     58189 non-null  int64
 33  ProductLine           58189 non-null  object
 34  ModelName             58189 non-null  object
 35  Photo                 58189 non-null  object
 36  ProductDescription    58189 non-null  object
 37  StartDate             58189 non-null  datetime64[ns]
 38  FirstName             58189 non-null  object
 39  LastName              58189 non-null  object
 40  FullName              58189 non-null  object
 41  BirthDate             58189 non-null  datetime64[ns]
 42  MaritalStatus         58189 non-null  object
 43  Gender                58189 non-null  object
 44  YearlyIncome          58189 non-null  int64
 45  TotalChildren         58189 non-null  int64
 46  NumberChildrenAtHome  58189 non-null  int64
 47  Education             58189 non-null  object
 48  Occupation            58189 non-null  object
 49  HouseOwnerFlag        58189 non-null  int64
 50  NumberCarsOwned       58189 non-null  int64
 51  AddressLine1          58189 non-null  object
 52  DateFirstPurchase     58189 non-null  datetime64[ns]
 53  CommuteDistance       58189 non-null  object
 54  Region                58189 non-null  object
 55  Country               58189 non-null  object
 56  Group                 58189 non-null  object
 57  RegionImage           58189 non-null  object
```

```
dtypes: datetime64[ns](5), float64(16), int64(10), object(27)
memory usage: 25.7+ MB
```

In [246…
```python
# Check shape of the data after merging

print(f"Number of Rows: {df.shape[0]}")
print(f"Number of Columns: {df.shape[1]} \n")
```

```
Number of Rows: 58189
Number of Columns: 58
```

In [247…
```python
df.describe().transpose()
```

Out[247]:

	count	mean	min	25%	50%	
OrderDate	58189	2016-06-03 03:56:09.605939200	2014-01-01 00:00:00	2016-04-01 00:00:00	2016-07-07 00:00:00	201 00:
ShipDate	58189	2016-06-10 04:03:24.657237760	2014-01-08 00:00:00	2016-04-08 00:00:00	2016-07-14 00:00:00	201 00:
SalesOrderLineNumber	58189.0	1.887453	1.0	1.0	2.0	
OrderQuantity	58189.0	1.569386	1.0	1.0	1.0	
UnitPrice	58189.0	413.888218	0.5725	4.99	24.49	26
TotalProductCost	58189.0	296.539185	0.8565	3.3623	12.1924	343
SalesAmount	58189.0	503.66627	2.29	8.99	32.6	5
TaxAmt	58189.0	40.293303	0.1832	0.7192	2.608	43
Unnamed: 13	0.0	NaN	NaN	NaN	NaN	
Unnamed: 14	0.0	NaN	NaN	NaN	NaN	
Unnamed: 15	58189.0	503.666269	2.29	8.99	32.6	5
Unnamed: 16	58189.0	0.000001	0.0	0.0	0.0	
Unnamed: 17	0.0	NaN	NaN	NaN	NaN	
Unnamed: 18	58189.0	38.398254	-5106.9068	1.4335	6.2537	21
Unnamed: 19	0.0	NaN	NaN	NaN	NaN	
StandardCost_x	58189.0	296.539185	0.8565	3.3623	12.1924	343
List Price	58189.0	503.66627	2.29	8.99	32.6	5
Unnamed: 22	0.0	NaN	NaN	NaN	NaN	
diif std cost	58189.0	0.0	0.0	0.0	0.0	
diff list price	58189.0	0.0	0.0	0.0	0.0	
StandardCost_y	58189.0	296.539185	0.8565	3.3623	12.1924	343
ListPrice	58189.0	503.66627	2.29	8.99	32.6	5
DaysToManufacture	58189.0	1.045215	0.0	0.0	0.0	
StartDate	58189	2007-05-14 02:44:51.848974848	2005-07-01 00:00:00	2007-07-01 00:00:00	2007-07-01 00:00:00	200 00:
BirthDate	58189	1962-03-02 12:33:19.305710720	1910-08-13 00:00:00	1954-12-20 00:00:00	1963-09-19 00:00:00	197 00:
YearlyIncome	58189.0	59769.887779	10000.0	30000.0	60000.0	80
TotalChildren	58189.0	1.838921	0.0	0.0	2.0	
NumberChildrenAtHome	58189.0	1.073502	0.0	0.0	0.0	
HouseOwnerFlag	58189.0	0.69056	0.0	0.0	1.0	
NumberCarsOwned	58189.0	1.502466	0.0	1.0	2.0	
DateFirstPurchase	58189	2015-12-23 02:50:33.356820224	2014-01-01 00:00:00	2015-06-21 00:00:00	2016-03-12 00:00:00	201 00:

In [248…
```
# Check for duplicate data
df.duplicated().sum()
```

Out[248]:
```
0
```

In [249…
```
### Handling missing data

def missing_pct(df):
    # Calculate missing value and their percentage for each column
    missing_count_percent = df.isnull().sum() * 100 / df.shape[0]
    df_missing_count_percent = pd.DataFrame(missing_count_percent).round(2)
    df_missing_count_percent = df_missing_count_percent.reset_index().renam
                    columns={
                            'index':'Column',
                            0:'Missing_Percentage (%)'
                    }
                )
    df_missing_value = df.isnull().sum()
    df_missing_value = df_missing_value.reset_index().rename(
                    columns={
                            'index':'Column',
                            0:'Missing_value_count'
                    }
                )
    # Sort the data frame
    #df_missing = df_missing.sort_values('Missing_Percentage (%)', ascendin
    Final = df_missing_value.merge(df_missing_count_percent, how = 'inner',
    Final = Final.sort_values(by = 'Missing_Percentage (%)',ascending = Fal
    return Final
```

In [250…
```
# Applying the custom function

missing_pct(df)
```

Out[250]:

	Column	Missing_value_count	Missing_Percentage (%)
22	Unnamed: 22	58189	100.00
19	Unnamed: 19	58189	100.00
14	Unnamed: 14	58189	100.00
13	Unnamed: 13	58189	100.00
17	Unnamed: 17	58189	100.00
30	Color	27442	47.16
0	ProductKey	0	0.00
42	MaritalStatus	0	0.00
41	BirthDate	0	0.00
39	LastName	0	0.00
40	FullName	0	0.00
38	FirstName	0	0.00
37	StartDate	0	0.00
36	ProductDescription	0	0.00
35	Photo	0	0.00
34	ModelName	0	0.00
43	Gender	0	0.00
44	YearlyIncome	0	0.00
32	DaysToManufacture	0	0.00
45	TotalChildren	0	0.00
46	NumberChildrenAtHome	0	0.00
47	Education	0	0.00
48	Occupation	0	0.00
49	HouseOwnerFlag	0	0.00
50	NumberCarsOwned	0	0.00
51	AddressLine1	0	0.00
52	DateFirstPurchase	0	0.00
53	CommuteDistance	0	0.00
54	Region	0	0.00
55	Country	0	0.00
56	Group	0	0.00
33	ProductLine	0	0.00
29	StandardCost_y	0	0.00
31	ListPrice	0	0.00
12	TaxAmt	0	0.00
2	ShipDate	0	0.00
3	CustomerKey	0	0.00

	Column	Missing_value_count	Missing_Percentage (%)
4	PromotionKey	0	0.00
5	SalesTerritoryKey	0	0.00
6	SalesOrderNumber	0	0.00
7	SalesOrderLineNumber	0	0.00
8	OrderQuantity	0	0.00
9	UnitPrice	0	0.00
10	TotalProductCost	0	0.00
11	SalesAmount	0	0.00
15	Unnamed: 15	0	0.00
1	OrderDate	0	0.00
16	Unnamed: 16	0	0.00
18	Unnamed: 18	0	0.00
20	StandardCost_x	0	0.00
21	List Price	0	0.00
23	diif std cost	0	0.00
24	diff list price	0	0.00
25	DateKey	0	0.00
26	ProductName	0	0.00
27	SubCategory	0	0.00
28	Category	0	0.00
57	RegionImage	0	0.00

In [251…
```
#  Drop columns with nan values

df= df.dropna(axis=1)
```

In [252…
```
### Adding columns

# Extracting Year from OrderDate
df['sale_year'] = df['OrderDate'].dt.year

# Extracting Month from OrderDate
df['sale_month'] = df['OrderDate'].dt.month

# Extracting day from OrderDate
df['sale_day'] = df['OrderDate'].dt.day

# Extracting dayofweek from OrderDate
df['sale_week'] = df['OrderDate'].dt.dayofweek

# Extracting day_name from OrderDate
df['sale_day_name'] = df['OrderDate'].dt.day_name()

# Extracting Month Year from OrderDate
df['year_month'] = df['OrderDate'].apply(lambda x:x.strftime('%Y-%m'))

# Calculate Total Invoice Amount
df['total_Invoice_amount'] = df['SalesAmount'] + df['TaxAmt']
```

```
# Considering only salesamount and total_sales_amount to calculate profit
df['profit'] = (df['UnitPrice']*df['OrderQuantity']) - df['TotalProductCost

# Removing extra character from the string
df['ProductName'] = df['ProductName'].str.replace(',','-')

# Calculate Age
df['Age'] = df['OrderDate'].dt.year - df['BirthDate'].dt.year
```

Exploring data

Basic Overview

In [255…

```
#### List of product's category

df['Category'].unique().tolist()
```

Out[255]:

```
['Bikes', 'Accessories', 'Clothing']
```

In [256…

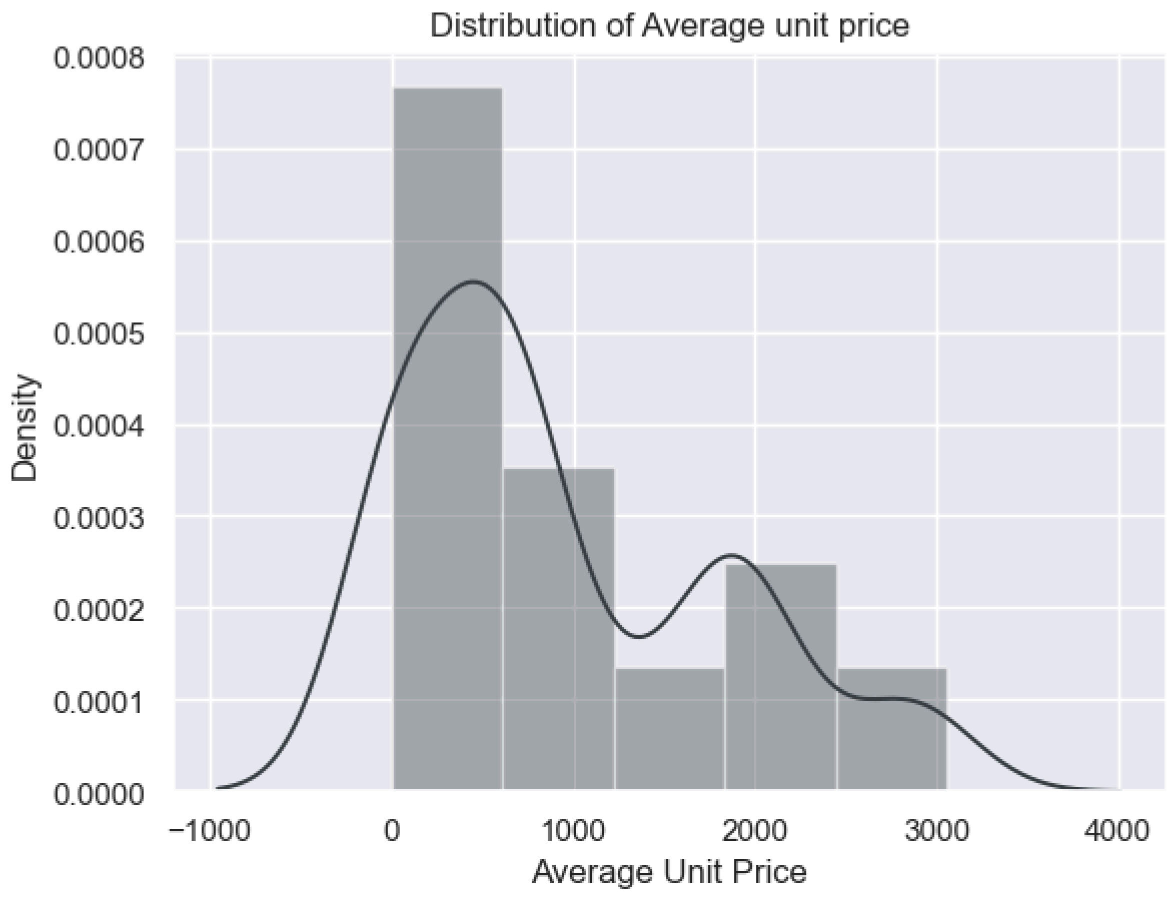
```
#### List of product's subcategory

df['SubCategory'].unique().tolist()
```

Out[256]:

```
['Road Bikes',
 'Mountain Bikes',
 'Bottles and Cages',
 'Gloves',
 'Tires and Tubes',
 'Helmets',
 'Touring Bikes',
 'Jerseys',
 'Cleaners',
 'Caps',
 'Hydration Packs',
 'Socks',
 'Fenders',
 'Vests',
 'Bike Racks',
 'Bike Stands',
 'Shorts']
```

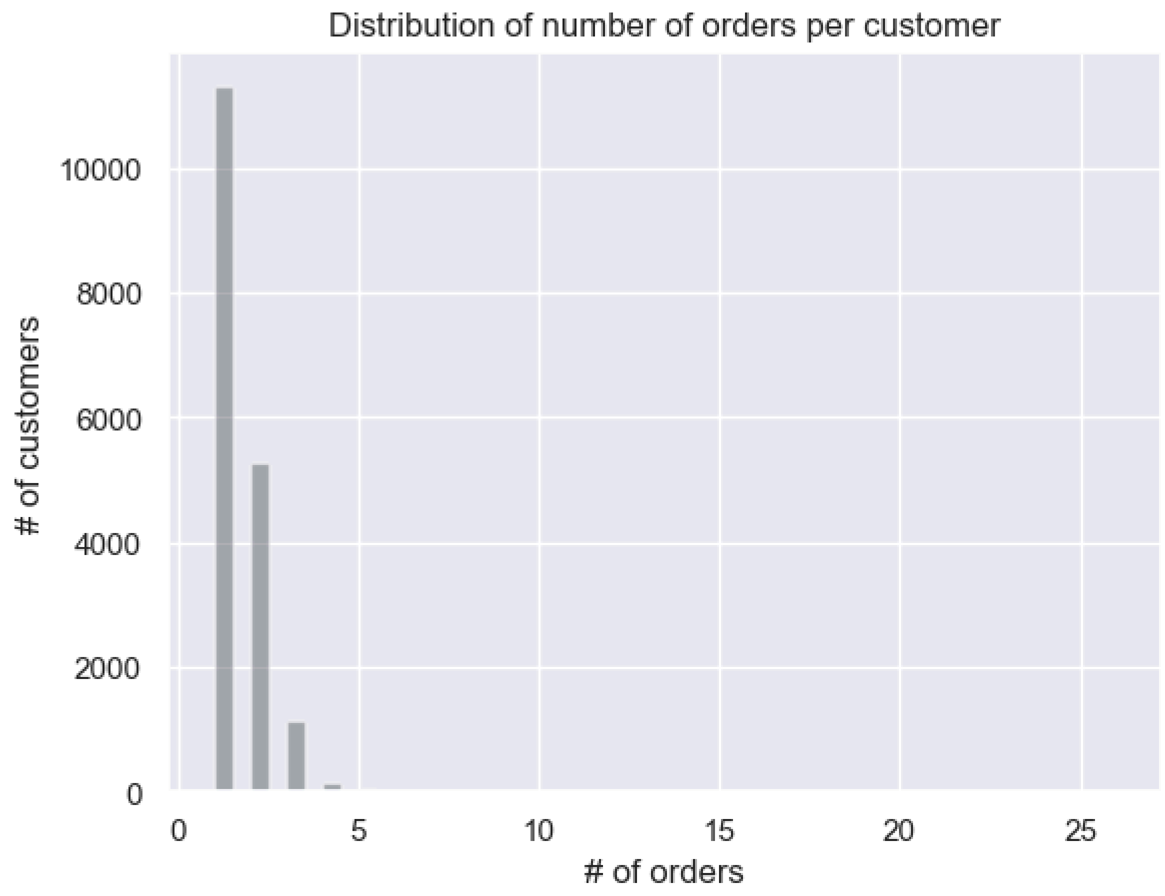
In [257…

```
#### Analysing UnitPrice

Avg_unit_price = df.groupby(['ProductKey'])['UnitPrice'].mean()
ax = sns.distplot(Avg_unit_price, kde=True, hist=True, color='#374045')
ax.set(title='Distribution of Average unit price',
       xlabel='Average Unit Price');
```



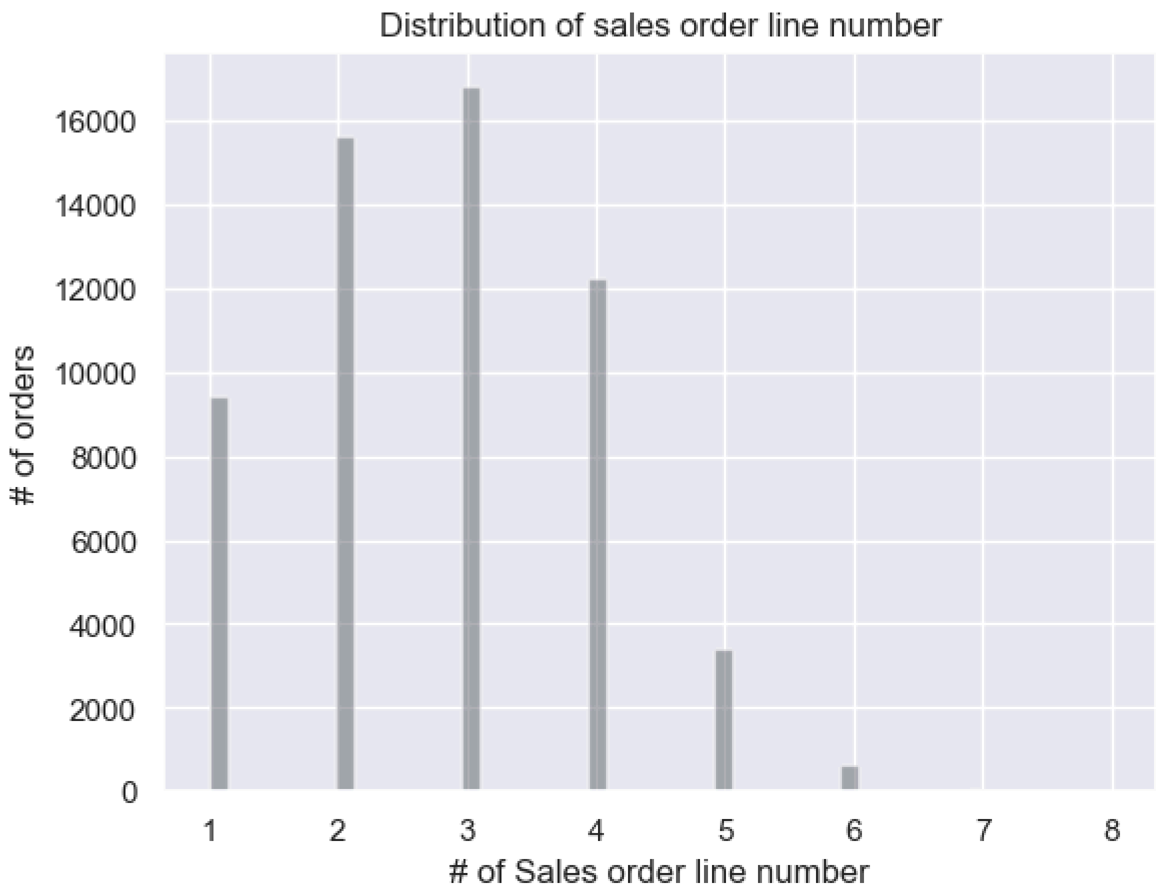
- Maximum of the product unit price is below $1000

In [259…
```
#### Sales order number distribution

n_orders = df.groupby(['CustomerKey'])['SalesOrderNumber'].nunique()
multi_orders_perc = np.sum(n_orders > 1)/df['CustomerKey'].nunique()
print(f"{100*multi_orders_perc:.2f}% of customers ordered more than once.")
```
```
36.97% of customers ordered more than once.
```
In [260…
```
ax = sns.distplot(n_orders, kde=False, color='#374045')
ax.set(title='Distribution of number of orders per customer',
       xlabel='# of orders',
       ylabel='# of customers');
```



In [261…

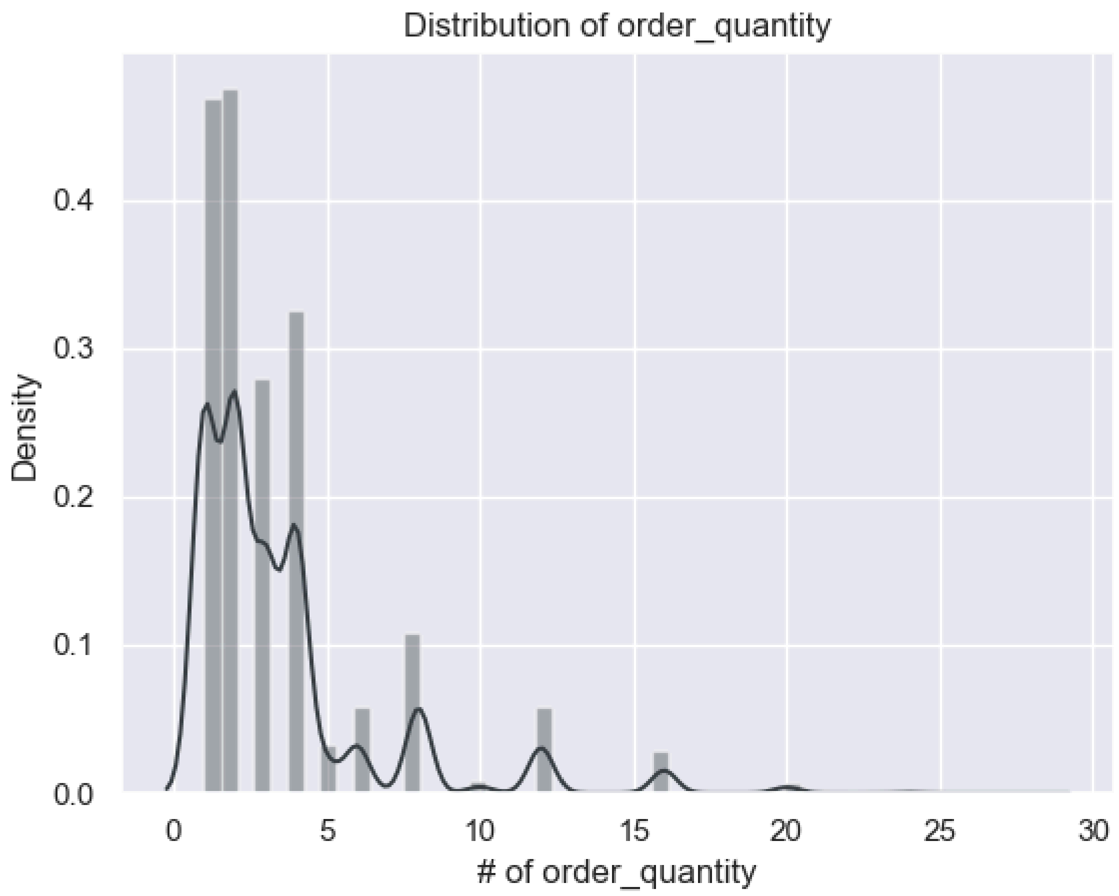
```
#### Sales order line number distribution

n_salesordernumber = df.groupby(['SalesOrderNumber'])['SalesOrderLineNumber
ax = sns.distplot(n_salesordernumber, kde=False, color='#374045')
ax.set(title='Distribution of sales order line number',
       xlabel='# of Sales order line number',
       ylabel='# of orders');
```



- Most of the time **three to two** products are ordered in a single order

In [263…

```
#### Sales Order Quantity distribution

n_order_quantity = df.groupby(['SalesOrderNumber'])['OrderQuantity'].sum()
ax = sns.distplot(n_order_quantity, kde=True, hist=True,color='#374045')
ax.set(title='Distribution of order_quantity',
       xlabel='# of order_quantity',
       );
```



- maximum quantity ordered for a product is below 5

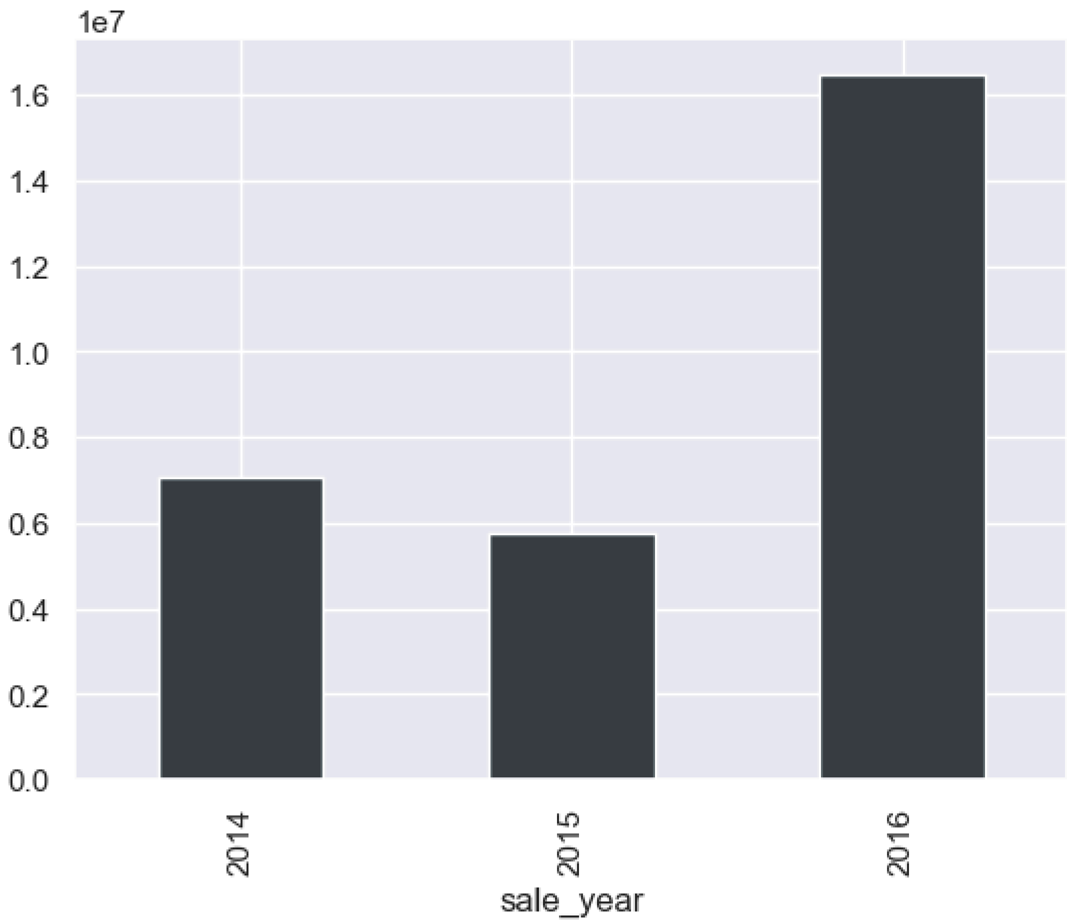
In [265…

```
#### Age Distribution


bins = [18, 30, 40, 50, 60, 70, 120]
labels = ['18-29', '30-39', '40-49', '50-59', '60-69', '70+']
df['agerange'] = pd.cut(df.Age, bins, labels = labels,include_lowest = True

age_distribution = df['agerange'].value_counts().to_frame().reset_index()

age_distribution.columns = ['Age Range','Population count']

fig = px.bar(age_distribution, x='Age Range', y='Population count', color_d
fig.update_layout(
    autosize=True,
    width=500,
    height=500,
    font=dict(size=10))
fig.show()
```

- A sizable portion of the clientele is made up of people between the ages of **40 and 59**.

Sales

In [268…
```
#### Year wise sales

df.groupby('sale_year')['SalesAmount'].sum().plot(kind='bar', color='#37404
```



- The year 2016 saw an exponential surge in sales

In [270…

```
#### Top 5 Selling Product

top_selling_product = df.groupby(['Category', 'SubCategory', 'ProductName']
top_selling_product
```

Out[270]:

Category	SubCategory	ProductName	OrderQuantity
Accessories	**Bottles and Cages**	**Water Bottle - 30 oz.**	6370
	Tires and Tubes	**Patch Kit/8 Patches**	4705
		Mountain Tire Tube	4551
		Road Tire Tube	3544
	Helmets	**Sport-100 Helmet- Red**	3398

In [271…

```
top_selling_product.reset_index(inplace=True)
fig = px.bar(top_selling_product, x='ProductName', y='OrderQuantity',color_
fig.update_layout(
    autosize=True,
    width=500,
    height=300,
    margin=dict(
        l=25,
        r=25,
        b=10,
        t=10,
    ),
    font=dict(size=8))
fig.show()
```

In [272…

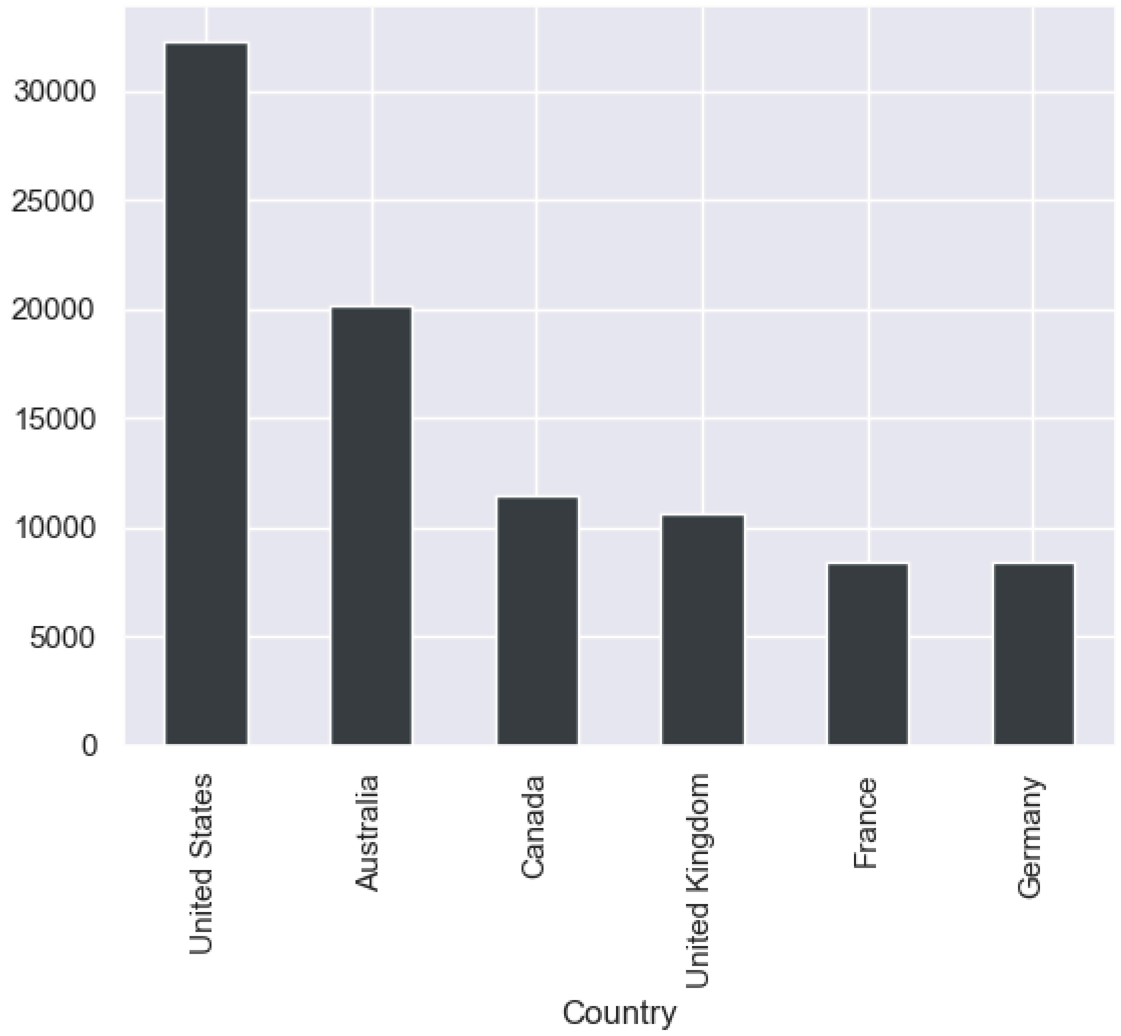
```
#### Quantity ordered based on category and subcategory from 2014 to 2016

cat_subcat_qty = df.groupby(['sale_year','Category', 'SubCategory'])['Order(
cat_subcat_qty = cat_subcat_qty.sort_values(['sale_year', 'Category'], ascer
cat_subcat_qty.style.bar(subset=['OrderQuantity'], color='#D9B300')
```

Out[272]:

sale_year	Category	SubCategory	OrderQuantity
2014	Bikes	Mountain Bikes	616
		Road Bikes	2876
2015	Bikes	Mountain Bikes	1661
		Road Bikes	3284
2016	Accessories	Bike Racks	493
		Bike Stands	394
		Bottles and Cages	12055
		Cleaners	1381
		Fenders	3239
		Helmets	9685
		Hydration Packs	1124
		Tires and Tubes	25518
	Bikes	Mountain Bikes	5490
		Road Bikes	6535
		Touring Bikes	3410
	Clothing	Caps	3178
		Gloves	2143
		Jerseys	5068
		Shorts	1491
		Socks	856
		Vests	824

In [273…

```
#### Country wise quantity ordered

country_qty_sales = df.groupby('Country')['OrderQuantity'].sum().sort_value
country_qty_sales.plot(kind='bar', color='#374045');
```



- High quantity of products is ordered from **Australia and United States**

Profit

In [276…

```
#### Overall profit based on order year, category and subcategory

cat_subcat_profit = df.groupby(['sale_year','Category', 'SubCategory'])['pro

#Sorting the results
cat_subcat_profit = cat_subcat_profit.sort_values(['sale_year', 'Category'],
cat_subcat_profit.style.bar(subset=['profit'], color='#D9B300')
```

Out[276]:

sale_year	Category	SubCategory	profit
2014	Bikes	Mountain Bikes	586874.557600
		Road Bikes	2256280.998300
2015	Bikes	Mountain Bikes	1019388.334900
		Road Bikes	1375064.915000
2016	Accessories	Bike Racks	23136.960000
		Bike Stands	23689.092000
		Bottles and Cages	34448.978300
		Cleaners	4299.868800
		Fenders	27711.633000
		Helmets	135167.732700
		Hydration Packs	24303.132200
		Tires and Tubes	144793.083200
	Bikes	Mountain Bikes	2907361.198000
		Road Bikes	1905953.736400
		Touring Bikes	1454872.695900
	Clothing	Caps	4331.831500
		Gloves	20895.744100
		Jerseys	37965.228300
		Shorts	41973.524600
		Socks	3055.841100
		Vests	20948.777000

- Major Profit is contributed by the Bike Category

In [278…

```
#### Low profit contributing product

df.groupby(['Category', 'SubCategory','ProductName'])['profit'].sum().nsmal
```

Out[278]:

Category	SubCategory	ProductName	profit
Clothing	Socks	Racing Socks- L	1474.4574
		Racing Socks- M	1581.3837
Accessories	Cleaners	Bike Wash - Dissolver	4299.8688
	Tires and Tubes	Patch Kit/8 Patches	4314.8350
Clothing	Caps	AWC Logo Cap	4331.8315
Accessories	Tires and Tubes	Touring Tire Tube	4363.8089
Clothing	Jerseys	Long-Sleeve Logo Jersey- XL	4495.6007
		Short-Sleeve Classic Jersey- L	4544.8782
		Long-Sleeve Logo Jersey- S	4610.5777
		Short-Sleeve Classic Jersey- M	4793.2322

In [279…

```
#### Profitability by country

country_sales = df.groupby('Country')[
    ['SalesAmount', 'profit']  # only select numeric columns here
].sum()

country_sales.reset_index(inplace=True)

fig = px.bar(
    country_sales,
    x='Country',
    y=['SalesAmount', 'profit'],
    barmode='group',
    title="Sales and Profit by Country",
    height=400
)
fig.show()
```

- High volume of profit is earned from **Australia and United States**

Question and Answers

In [282…

```
#### How efficient are the logistics?

# Adding manufacturing days to the order received date
df['OrderreadyDate'] = df['OrderDate'] + pd.to_timedelta(df['DaysToManufactu

# Check the delay between order shipment date and order ready to supply
df['shipping_efficiency'] = (df['ShipDate'] - df['OrderreadyDate']).dt.days

fig = px.histogram(df, x="shipping_efficiency", color_discrete_sequence=['#3
fig.update_layout(
    autosize=True,
    width=300,
    height=300,
    margin=dict(
        l=25,
        r=25,
        b=10,
        t=10,
    ),
    font=dict(size=10))
fig.show()
```

- The average order has a gap of 7 days between the day the order is ready for export from the factory and the date it was shipped
- Management must work to reduce this gap toward 3 days.

In [284…
```python
### What was the best month for sales? How much was earned that month ?

# Ensure you only sum numeric columns, explicitly selecting them
month_sales = df.groupby('sale_month')[['SalesAmount', 'profit']].sum()

# Reset the index to turn 'sale_month' back into a column
month_sales.reset_index(inplace=True)

# Plot the bar chart
fig = px.bar(
    month_sales,
    x='sale_month',
    y='SalesAmount',
    text_auto='.2s',
    hover_data=['sale_month', 'SalesAmount'],
    color='profit',
    height=400
)

fig.show()
```

- There are large profit transactions in the months of **June, November, and December**

In [287…

```
## What time should we display advertisement to maximize likelihood of custo

sales_by_week = df.groupby(['sale_day_name']).count()['SalesAmount'].reset_i

fig = px.line(sales_by_week, x='sale_day_name', y='SalesAmount', title='Sale
fig.update_layout(
    autosize=True,
    width=300,
    height=300,
    margin=dict(
        l=25,
        r=25,
        b=10,
        t=10,
    ),
    font=dict(size=7))
fig.show()
```

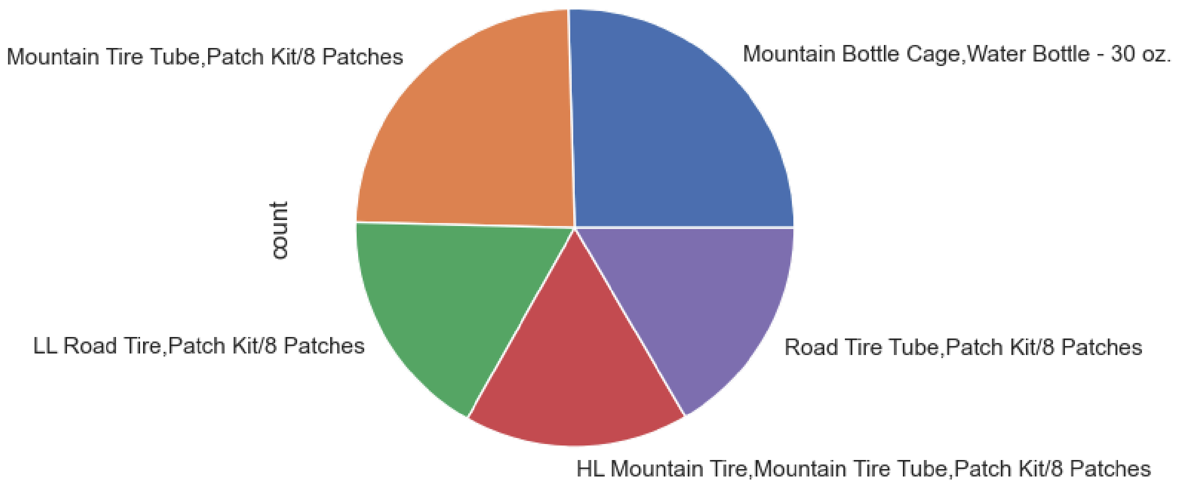
- High sales orders are seen on **Wednesday and Saturday**, therefore we can promote our product during these workweek

Which products are most often sold together?

In [290…
```
# By setting keep on False, all duplicates are True since we only want repea
dup_order = df[df['SalesOrderNumber'].duplicated(keep=False)]
```

In [291…
```
# Group the data based on sales order number and product name because the pr
# that bought together will have share same order number
dup_order['grouped'] = df.groupby('SalesOrderNumber')['ProductName'].transfo
dup_order = dup_order[['SalesOrderNumber', 'grouped']].drop_duplicates()
```

In [292…
```
count = dup_order['grouped'].value_counts()[0:5].plot.pie()
```



- From the above pie diagram we can draw a conclusion that these products are mostly Purchased together

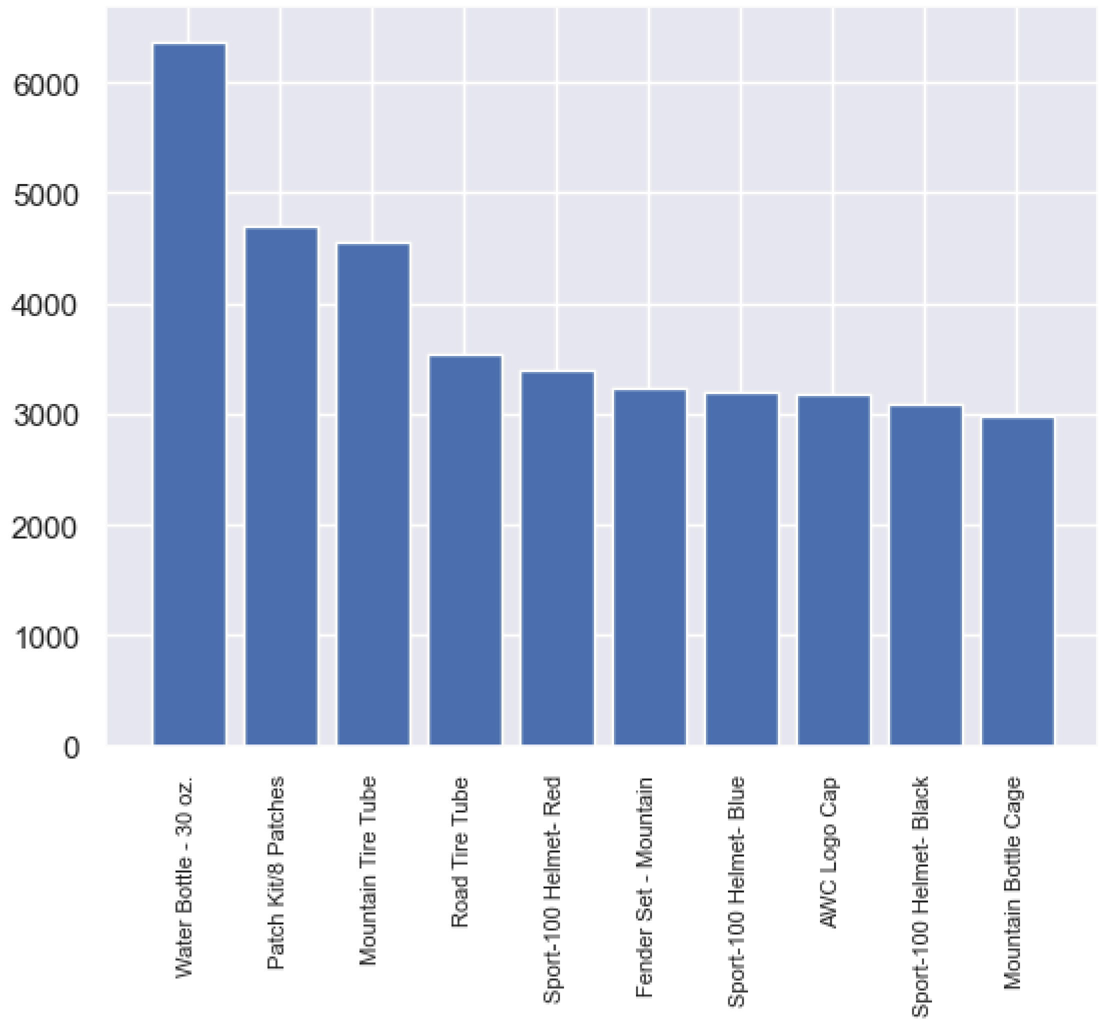
In [294…
```
count = Counter()

for row in dup_order['grouped']:
    row_list  = row.split(',')
    count.update(Counter(combinations(row_list, 2)))

for key, value in count.most_common(10):
    print(key, value)
```

```
('Mountain Bottle Cage', 'Water Bottle - 30 oz.') 1623
('Road Bottle Cage', 'Water Bottle - 30 oz.') 1513
('HL Mountain Tire', 'Mountain Tire Tube') 915
('Touring Tire', 'Touring Tire Tube') 758
('Mountain Tire Tube', 'Patch Kit/8 Patches') 737
('Mountain Tire Tube', 'ML Mountain Tire') 727
('Water Bottle - 30 oz.', 'AWC Logo Cap') 599
('Road Tire Tube', 'ML Road Tire') 580
('Road Tire Tube', 'Patch Kit/8 Patches') 556
('HL Road Tire', 'Road Tire Tube') 552
```

- The above product can be sold in a bundle or a combined package for discount

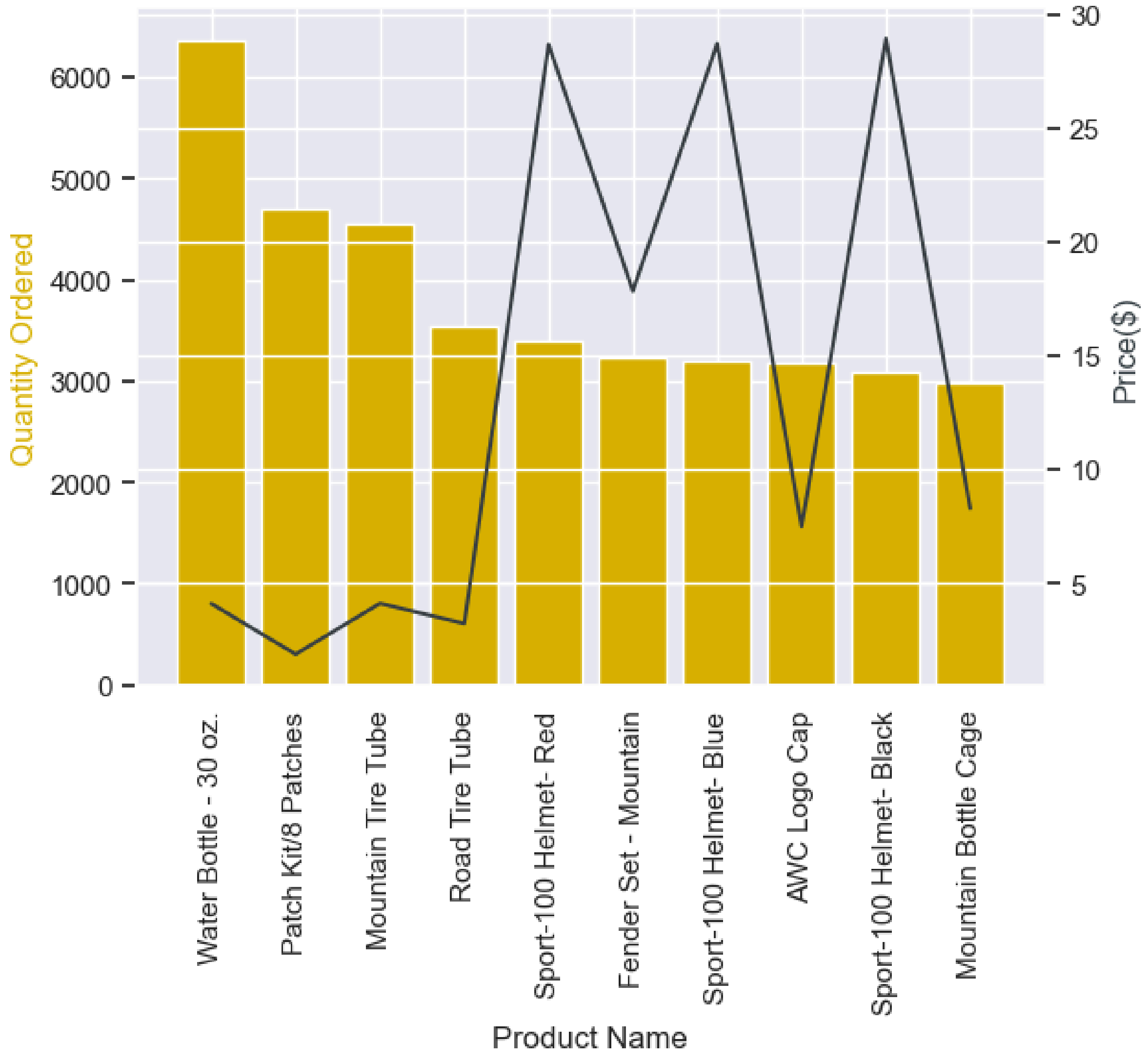
In [296…

```
# Which product sold the most? why do you think it sold the most?

product_group = df.groupby('ProductName')
quantity_ordered = product_group['OrderQuantity'].sum().sort_values(ascendi
products = quantity_ordered.index.tolist()

plt.bar(products, quantity_ordered, )
plt.xticks(products, rotation='vertical', size=8)
plt.show()
```



In [297…
```
prices = df.groupby('ProductName')['UnitPrice'].mean()
prices = prices[products]
```

In [298…
```
fig, ax1 = plt.subplots()

ax2 = ax1.twinx()
ax1.bar(products, quantity_ordered, color='#D9B300')
ax2.plot(products, prices, '#374045')

ax1.set_xlabel('Product Name')
ax1.set_ylabel('Quantity Ordered', color='#D9B300')
ax2.set_ylabel('Price($)', color='#374045')
ax1.set_xticklabels(products, rotation='vertical')

plt.show();
```



In [299…
```
prices.corr(quantity_ordered)
```

Out[299]:
```
-0.5333019792658484
```

- There is a **high negative correlation** between **Price and number of Quantity ordered**
- we can conclude that **low price product has high demand**

```
In [302…  # Compare most ordered product by gender

          male = df[df["Gender"]=="M"]
          female = df[df["Gender"]=="F"]
```

```
In [303…  male_ord_qty = male.groupby(['ProductName'],as_index=False)['OrderQuantity']
          male_ord_qty.columns=['ProductName','Order_Qty_Male']

          female_ord_qty = female.groupby(['ProductName'],as_index=False)['OrderQuant
          female_ord_qty.columns=['ProductName','Order_Qty_Female']

          df_merge = pd.merge(male_ord_qty, female_ord_qty, on='ProductName')
```

```
In [304…  fig = px.line(df_merge, x="ProductName", y=["Order_Qty_Male","Order_Qty_Fema
          fig.update_layout(
              autosize=True,
              width=800,
              height=400)
          fig.show()
```

```
In [306…  # Does Gender and home ownership matter in order purchasing

          fig = px.imshow(df.groupby(["Gender", "HouseOwnerFlag"])["SalesAmount"].mear
                          labels=dict(color="Average Purchase"))
          fig.show()
```

- It's interesting to note that the average amount spent by men without permanent addresses is low, whilst the average amount spent by women without permanent addresses is higher.

In [309…

```
# Number of childer and Purchase correlation

df_1 = df.groupby(["NumberChildrenAtHome"])["SalesAmount"].mean().to_frame(
df_1.reset_index(inplace=True)
fig = px.bar(df_1, x='NumberChildrenAtHome', y='SalesAmount',color_discrete_
fig.update_layout(
    autosize=False,
    width=300,
    height=300,
    margin=dict(
        l=25,
        r=25,
        b=10,
        t=10,
    ))
fig.show()
```

In [311…

```
# Education, Occupation and Purchase correlation

fig = px.imshow(df.groupby(["Education", "Occupation"])["SalesAmount"].mean
                labels=dict(color="Average Purchase"))
fig.show()
```

Maritial Status single and above 50 age purchase

In [313…
```
df_2 = df[(df['MaritalStatus']=='S')&(df['Age']>50)]
```

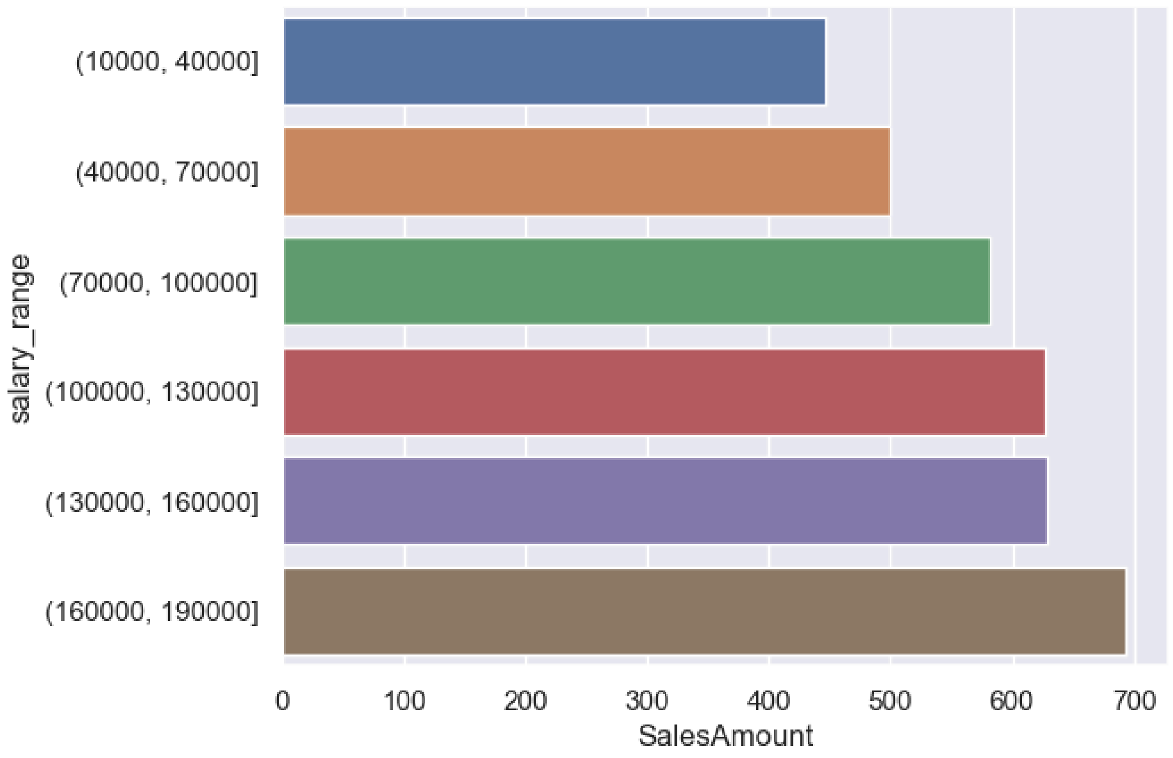
In [314…
```
df_2 = df_2.groupby('agerange')['SalesAmount'].mean().to_frame().dropna()
df_2.reset_index(inplace=True)
fig = px.bar(df_2, x='agerange', y='SalesAmount', color_discrete_sequence=[
fig.update_layout(
    autosize=False,
    width=300,
    height=300,
    margin=dict(
        l=25,
        r=25,
        b=10,
        t=10,
    ))
fig.show()
```

In [315…
```
# Which age group has produced the most revenue?

df_3 = df.groupby('agerange')['SalesAmount'].mean().to_frame().dropna()
df_3.reset_index(inplace=True)
fig = px.bar(df_3, x='agerange', y='SalesAmount', color_discrete_sequence=[
fig.update_layout(
    autosize=False,
    width=300,
    height=300,
    margin=dict(
        l=25,
        r=25,
        b=10,
        t=10,
    ))
fig.show()
```

In [317…
```
#  Yearly income range and purchase correlation

def create_bins(lower_bound, width, quantity):
    """ create_bins returns an equal-width (distance) partitioning.
        It returns an ascending list of tuples, representing the intervals.
        A tuple bins[i], i.e. (bins[i][0], bins[i][1])  with i > 0
        and i < quantity, satisfies the following conditions:
            (1) bins[i][0] + width == bins[i][1]
            (2) bins[i-1][0] + width == bins[i][0] and
                bins[i-1][1] + width == bins[i][1]
    """


    bins = []
    for low in range(lower_bound,
                     lower_bound + quantity*width + 1, width):
        bins.append((low, low+width))
    return bins
```

In [318…
```
bins = create_bins(lower_bound=10000,
                   width=30000,
                   quantity=5)
bins2 = pd.IntervalIndex.from_tuples(bins)
df['salary_range'] = pd.cut(df['YearlyIncome'], bins2)
```

In [319…
```
df_4 = df.groupby('salary_range')['SalesAmount'].mean().to_frame()
df_4.reset_index(inplace=True)
sns.barplot(x="SalesAmount", y="salary_range", data=df_4);
```



- High salary range leads to increase in purchase

Paritial high school vs bachlors income mean and most ordered product

```
In [322…  df_6 = df[(df['Education']=='Partial High School')|(df['Education']=='Bache
```

```
In [323…  df_6.reset_index(inplace=True)
          fig = px.bar(df_6, x='Education', y='YearlyIncome')
          fig.update_layout(
              autosize=False,
              width=300,
              height=300,
              margin=dict(
                  l=25,
                  r=25,
                  b=10,
                  t=10,
              ))
          fig.show()
```

In [324…

```
df_7 = df[(df['Education']=='Partial High School')|(df['Education']=='Bache
df_7 = df_7.groupby(['Education','ProductName'])['OrderQuantity'].mean().to_
df_7.reset_index(inplace=True)
fig = px.bar(df_7, x="Education",
             y="OrderQuantity", color="ProductName",
             title="Paritial high school vs bachlors expense analysis",
             barmode="group")
fig.show()
```

- Customers with a **high school diploma and modest annual income buy more products** than people with bachelor's degrees

Customer Segmentation

In [327…
```
# RFM stands for recency, frequency, monetary value.
#  In business analytics, we often use this concept to divide
#  customers into different segments, like high-value customers,
#  medium value customers or low-value customers, and similarly many others.
```

In [328…
```
# Recency: How recently has the customer made a transaction with us
# Frequency: How frequent is the customer in ordering/buying some product fr
# Monetary: How much does the customer spend on purchasing products from us
```

In [329…
```
#  calculating recency for customers who had made a purchase with a company

df_recency = df.groupby(by='FullName',
                        as_index=False)['OrderDate'].max()
df_recency.columns = ['CustomerName', 'LastPurchaseDate']
recent_date = df_recency['LastPurchaseDate'].max()
df_recency['Recency'] = df_recency['LastPurchaseDate'].apply(
    lambda x: (recent_date - x).days)
```

In [330…
```
#  calculating the frequency of frequent transactions of the
#  customer in ordering/buying some product from the company.


frequency_df = df.drop_duplicates().groupby(
    by=['FullName'], as_index=False)['OrderDate'].count()
frequency_df.columns = ['CustomerName', 'Frequency']
# frequency_df.head()
```

In [331…
```
monetary_df = df.groupby(by='FullName', as_index=False)['SalesAmount'].sum(
monetary_df.columns = ['CustomerName', 'Monetary']
# monetary_df.head()
```

In [332…
```
# merging dataset
rf_df = df_recency.merge(frequency_df, on='CustomerName')
rfm_df = rf_df.merge(monetary_df, on='CustomerName').drop(
    columns='LastPurchaseDate')
# rfm_df.head()
```

In [333…
```
rfm_df['R_rank'] = rfm_df['Recency'].rank(ascending=False)
rfm_df['F_rank'] = rfm_df['Frequency'].rank(ascending=True)
rfm_df['M_rank'] = rfm_df['Monetary'].rank(ascending=True)

# normalizing the rank of the customers
rfm_df['R_rank_norm'] = (rfm_df['R_rank']/rfm_df['R_rank'].max())*100
rfm_df['F_rank_norm'] = (rfm_df['F_rank']/rfm_df['F_rank'].max())*100
rfm_df['M_rank_norm'] = (rfm_df['F_rank']/rfm_df['M_rank'].max())*100

rfm_df.drop(columns=['R_rank', 'F_rank', 'M_rank'], inplace=True)

# rfm_df.head()
```

In [334…
```
rfm_df['RFM_Score'] = 0.15*rfm_df['R_rank_norm']+0.28 * \
    rfm_df['F_rank_norm']+0.57*rfm_df['M_rank_norm']
rfm_df['RFM_Score'] *= 0.05
```

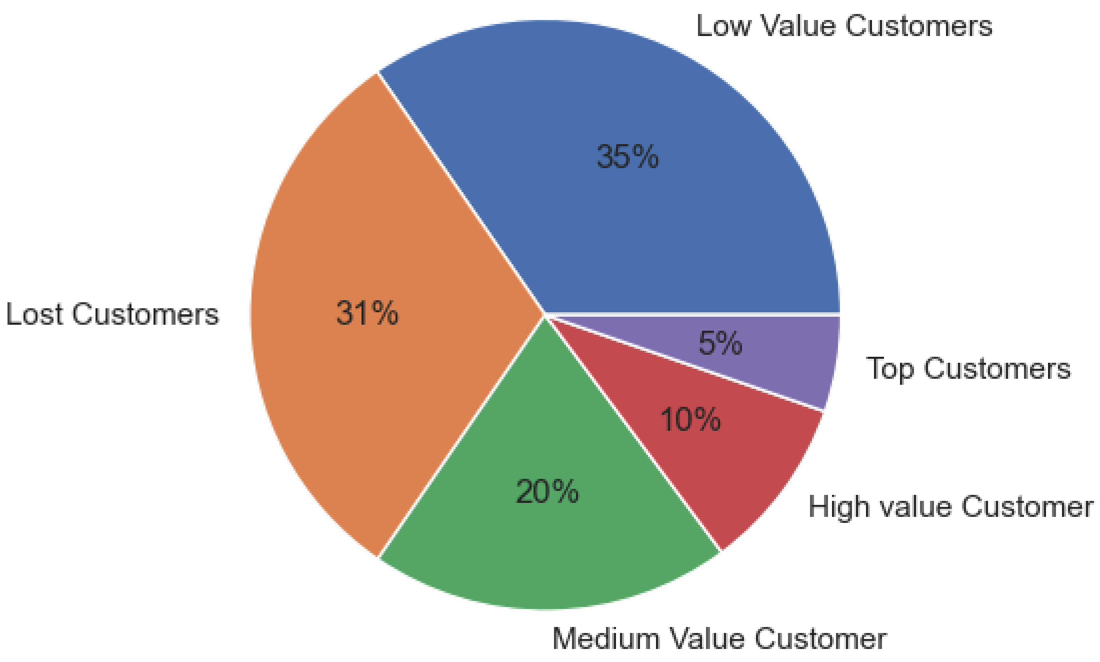
```
rfm_df = rfm_df.round(2)
# rfm_df[['CustomerName', 'RFM_Score']].head(7)
```

In [335…

```
rfm_df["Customer_segment"] = np.where(rfm_df['RFM_Score'] >
                                      4.5, "Top Customers",
                                      (np.where(
                                        rfm_df['RFM_Score'] > 4,
                                        "High value Customer",
                                        (np.where(
    rfm_df['RFM_Score'] > 3,
                             "Medium Value Customer",
                             np.where(rfm_df['RFM_Score'] > 1.6,
                            'Low Value Customers', 'Lost Customers'))))))
# rfm_df[['CustomerName', 'RFM_Score', 'Customer_segment']].head(20)
```

In [336…

```
plt.pie(rfm_df.Customer_segment.value_counts(),
        labels=rfm_df.Customer_segment.value_counts().index,
        autopct='%.0f%%')
plt.show()
```



- According to the customer segmentation described above, approximately **15% of our clients are high value clients**, whereas the **majority of our clientele are low value and lost clients**

Cohort Analysis

In [339…

```
# create an invoice month

# Function for month
def get_month(x):
  return dt.datetime(x.year, x.month,1)

# apply the function
df['InvoiceMonth'] = df['OrderDate'].apply(get_month)
```

```
# create a column index with the minimum invoice date aka first time custome
df['CohortMonth'] = df.groupby('CustomerKey')['InvoiceMonth'].transform('mi
```

In [340…

```
# create a date element function to get a series for subtranction
def get_date_elements(data,column):
  day = data[column].dt.day
  month = data[column].dt.month
  year = data[column].dt.year
  return day, month, year
```

In [341…

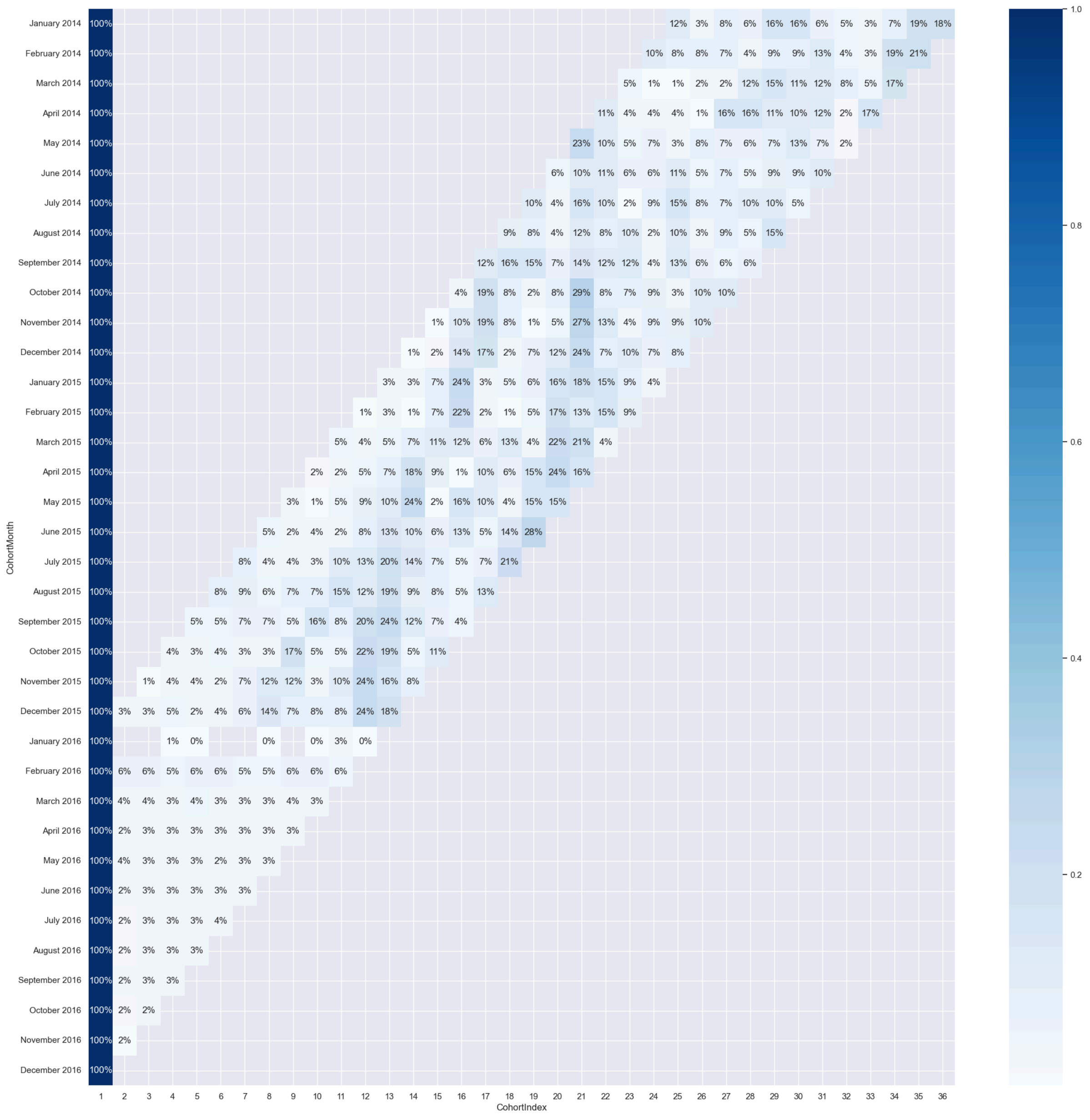
```
# get date elements for our cohort and invoice columns(one dimentional Serie
_, Invoice_month, Invoice_year = get_date_elements(df, 'InvoiceMonth')
_, Cohort_month, Cohort_year = get_date_elements(df, 'CohortMonth')

# create a cohort index
year_diff = Invoice_year - Cohort_year
month_diff = Invoice_month - Cohort_month
df['CohortIndex'] = year_diff*12+month_diff+1

# count the customer ID by grouping by Cohort Month and Cohort index
cohort_data = df.groupby(['CohortMonth','CohortIndex'])['CustomerKey'].apply

# create pivot table
cohort_table = cohort_data.pivot(index='CohortMonth', columns=['CohortIndex

# change index
cohort_table.index = cohort_table.index.strftime('%B %Y')

# cohort table for percentage
new_cohort_table = cohort_table.divide(cohort_table.iloc[:,0],axis=0)
```

In [342…

```
# create percentages
plt.figure(figsize=(25,25))
sns.heatmap(new_cohort_table, annot=True, cmap='Blues',fmt='.0%')
```

Out[342]:

```
<Axes: xlabel='CohortIndex', ylabel='CohortMonth'>
```



Thank you

```
In [ ]:
```