This Online Retail II data set contains all the transactions occurring for a UK-based and registered, non-store online retail between 01/12/2009 and 09/12/2011.The company mainly sells unique all-occasion gift-ware. Many customers of the company are wholesalers.

Attribute Information:

InvoiceNo: Invoice number. Nominal. A 6-digit integral number uniquely assigned to each transaction. If this code starts with the letter 'c', it indicates a cancellation.
StockCode: Product (item) code. Nominal. A 5-digit integral number uniquely assigned to each distinct product.
Description: Product (item) name. Nominal.
Quantity: The quantities of each product (item) per transaction. Numeric.
InvoiceDate: Invice date and time. Numeric. The day and time when a transaction was generated.
UnitPrice: Unit price. Numeric. Product price per unit in sterling (Â£).
CustomerID: Customer number. Nominal. A 5-digit integral number uniquely assigned to each customer.
Country: Country name. Nominal. The name of the country where a customer resides.

In [2]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import plotly
import plotly.express as px
```

In [3]:
```
df = pd.read_csv('customer_data.csv', encoding= 'unicode_escape')
```

In [4]:
```
# data is on an individual transaction level
df.head()
```

Out[4]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	12/1/2010 8:26	2.55	17850.0	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	12/1/2010 8:26	3.39	17850.0	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	12/1/2010 8:26	2.75	17850.0	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	12/1/2010 8:26	3.39	17850.0	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	12/1/2010 8:26	3.39	17850.0	United Kingdom

EDA

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 541909 entries, 0 to 541908
Data columns (total 8 columns):
 #   Column       Non-Null Count   Dtype  
---  ------       --------------   -----  
 0   InvoiceNo    541909 non-null  object 
 1   StockCode    541909 non-null  object 
 2   Description  540455 non-null  object 
 3   Quantity     541909 non-null  int64  
 4   InvoiceDate  541909 non-null  object 
 5   UnitPrice    541909 non-null  float64
 6   CustomerID   406829 non-null  float64
 7   Country      541909 non-null  object 
dtypes: float64(2), int64(1), object(5)
memory usage: 33.1+ MB
```

In [6]:
```
#negative quantities and price must be returned orders
df.describe()
```

Out[6]:

	Quantity	UnitPrice	CustomerID
count	541909.000000	541909.000000	406829.000000
mean	9.552250	4.611114	15287.690570
std	218.081158	96.759853	1713.600303
min	-80995.000000	-11062.060000	12346.000000
25%	1.000000	1.250000	13953.000000
50%	3.000000	2.080000	15152.000000
75%	10.000000	4.130000	16791.000000
max	80995.000000	38970.000000	18287.000000

In [7]:
```
df.shape
```

Out[7]: (541909, 8)

Starting to look into individual customer and product information

Total number of unique customers

In [73]:
```
df.head(1)
```

Out[73]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	Order_Value	Month
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	12/1/2010 8:26	2.55	17850.0	United Kingdom	15.3	12

In [9]:
```
len(df.CustomerID.unique())
```

Out[9]: 4373

Top 10 Customers Who Have Spent The Most Money (lifetime)

In [10]:
```
# create order value column

df['Order_Value'] = df.Quantity * df.UnitPrice
```

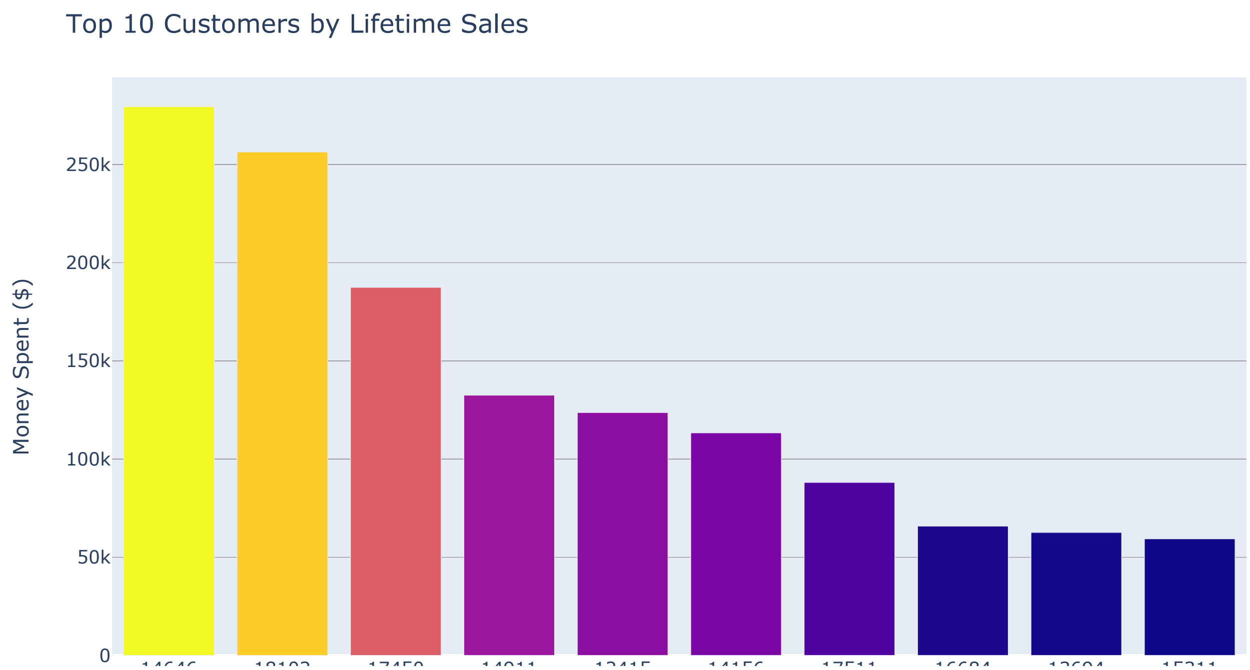
In [11]:
```
cust_sales = pd.DataFrame(df.groupby('CustomerID').Order_Value.sum()).reset_index()
top_10_customers_sales = cust_sales.sort_values('Order_Value', ascending=False)[:10]
```

In [12]:
```
top_10_customers_sales
```

Out[12]:

	CustomerID	Order_Value
1703	14646.0	279489.02
4233	18102.0	256438.49
3758	17450.0	187482.17
1895	14911.0	132572.62
55	12415.0	123725.45
1345	14156.0	113384.14
3801	17511.0	88125.38
3202	16684.0	65892.08
1005	13694.0	62653.10
2192	15311.0	59419.34

In [13]:
```
fig = px.bar(top_10_customers_sales, x = 'CustomerID' , y = 'Order_Value',
      hover_data=['CustomerID', 'Order_Value'], color='Order_Value',
      labels={'Order_Value':'Money Spent ($)', 'x':'Customer ID'})

fig.update_layout(xaxis_type='category',
                  title="Top 10 Customers by Lifetime Sales")
```



Unique products

In [14]:
```
len(df.InvoiceNo.unique())
```

Out[14]: 25900

In [16]:
```
prod_order_cnt = pd.DataFrame(df.groupby(['StockCode','Description']).InvoiceNo.count()).reset_index()

top_10_products = prod_order_cnt.sort_values('InvoiceNo', ascending=False)[:10]
top_10_products.rename(columns = {'InvoiceNo':'Num_Orders'}, inplace=True)

fig = px.bar(top_10_products, x='StockCode', y='Num_Orders',
            hover_data = ['StockCode','Description','Num_Orders'],
            text=top_10_products.Description)
fig.update_layout(xaxis_type='category',
                  title="Top 10 Products (lifetime unique orders)")
fig.show()
```

Orders per country

In [17]:
```
df.Country.unique()
```

Out[17]:
```
array(['United Kingdom', 'France', 'Australia', 'Netherlands', 'Germany',
       'Norway', 'EIRE', 'Switzerland', 'Spain', 'Poland', 'Portugal',
       'Italy', 'Belgium', 'Lithuania', 'Japan', 'Iceland',
       'Channel Islands', 'Denmark', 'Cyprus', 'Sweden', 'Austria',
       'Israel', 'Finland', 'Bahrain', 'Greece', 'Hong Kong', 'Singapore',
       'Lebanon', 'United Arab Emirates', 'Saudi Arabia',
       'Czech Republic', 'Canada', 'Unspecified', 'Brazil', 'USA',
       'European Community', 'Malta', 'RSA'], dtype=object)
```

In [18]:
```
df.head()
```

Out[18]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	Order_Value
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	12/1/2010 8:26	2.55	17850.0	United Kingdom	15.30
1	536365	71053	WHITE METAL LANTERN	6	12/1/2010 8:26	3.39	17850.0	United Kingdom	20.34
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	12/1/2010 8:26	2.75	17850.0	United Kingdom	22.00
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	12/1/2010 8:26	3.39	17850.0	United Kingdom	20.34
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	12/1/2010 8:26	3.39	17850.0	United Kingdom	20.34

In [19]:
```
from collections import Counter, OrderedDict
```

In [20]:
```
# Orders per country
orders_country = OrderedDict(Counter(df.Country))
orders_country
```

Out[20]:
```
OrderedDict([('United Kingdom', 495478),
             ('France', 8557),
             ('Australia', 1259),
             ('Netherlands', 2371),
             ('Germany', 9495),
             ('Norway', 1086),
             ('EIRE', 8196),
             ('Switzerland', 2002),
             ('Spain', 2533),
             ('Poland', 341),
             ('Portugal', 1519),
             ('Italy', 803),
             ('Belgium', 2069),
             ('Lithuania', 35),
             ('Japan', 358),
             ('Iceland', 182),
             ('Channel Islands', 758),
             ('Denmark', 389),
             ('Cyprus', 622),
             ('Sweden', 462),
             ('Austria', 401),
             ('Israel', 297),
             ('Finland', 695),
             ('Bahrain', 19),
             ('Greece', 146),
             ('Hong Kong', 288),
             ('Singapore', 229),
             ('Lebanon', 45),
             ('United Arab Emirates', 68),
             ('Saudi Arabia', 10),
             ('Czech Republic', 30),
             ('Canada', 151),
             ('Unspecified', 446),
             ('Brazil', 32),
             ('USA', 291),
             ('European Community', 61),
             ('Malta', 127),
             ('RSA', 58)])
```

Average purchase order value per customer

In [23]:
```
avg_order_value = pd.DataFrame(df.groupby('CustomerID').Order_Value.mean()).reset_index()
avg_order_value_purchases = avg_order_value.loc[avg_order_value.Order_Value > 0]

avg_order_value_purchases.head()
```

Out[23]:

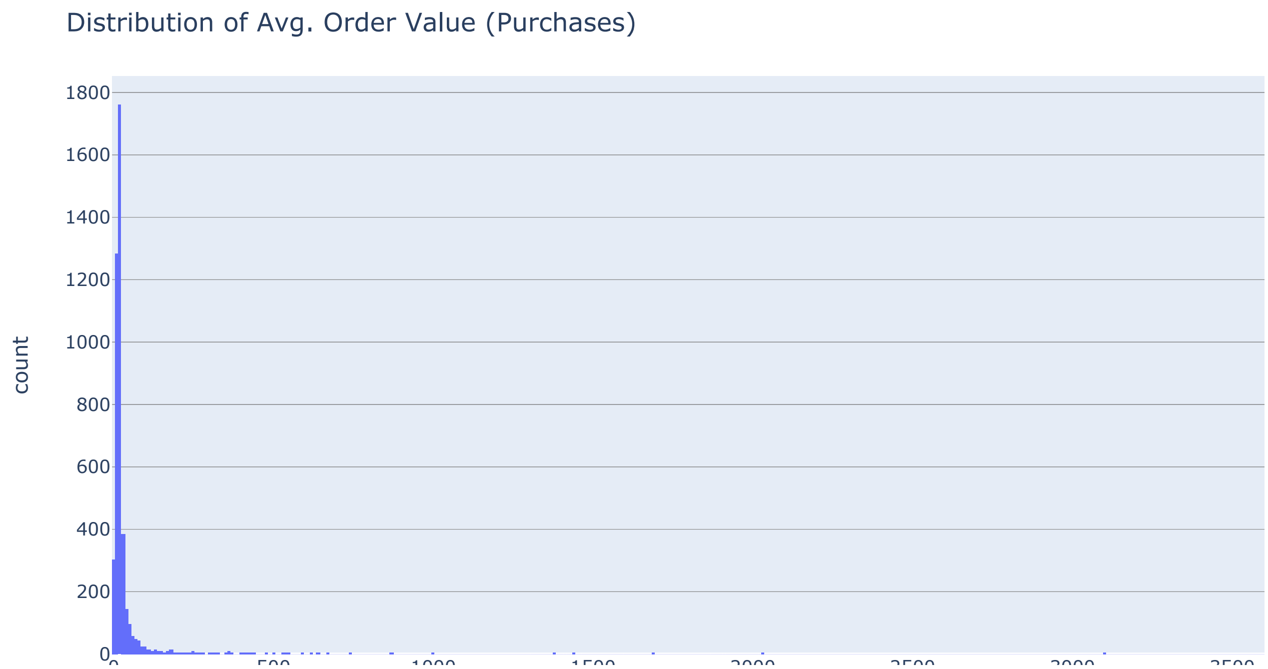
	CustomerID	Order_Value
1	12347.0	23.681319
2	12348.0	57.975484
3	12349.0	24.076027
4	12350.0	19.670588
5	12352.0	16.267474

In [24]:
```
avg_order_value_purchases.describe()
```

Out[24]:

	CustomerID	Order_Value
count	4322.000000	4.322000e+03
mean	15298.745025	3.125194e+01
std	1721.241678	1.049090e+02
min	12347.000000	5.652044e-16
25%	13812.250000	1.142488e+01
50%	15297.500000	1.701218e+01
75%	16777.750000	2.368220e+01
max	18287.000000	3.861000e+03

In [25]:
```
fig = px.histogram(avg_order_value_purchases, 'Order_Value')

fig.update_layout(title="Distribution of Avg. Order Value (Purchases)")
```



Getting data on a customer level

In order to segment the customers with a clustering algorithm, I need to get the data on a unique customer level.
These are the features I will create:

1) **average order value per customer**
2) **number of orders per customer**
3) **customer country**
4) **most frequently purchased product by customer**
5) **total spent by customer**
6) **most active month per customer**

In [74]:
```
df.head(3)
```

Out[74]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	Order_Value	Month
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	12/1/2010 8:26	2.55	17850.0	United Kingdom	15.30	12
1	536365	71053	WHITE METAL LANTERN	6	12/1/2010 8:26	3.39	17850.0	United Kingdom	20.34	12
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	12/1/2010 8:26	2.75	17850.0	United Kingdom	22.00	12

In [75]:
```
#1) average order value per customer

avg_ord_val = pd.DataFrame(df.groupby('CustomerID')['Order_Value'].mean()).reset_index()
avg_ord_val.rename(columns={'Order_Value':'Avg_Order_Value'}, inplace=True)
avg_ord_val
```

Out[75]:

	CustomerID	Avg_Order_Value
0	12346.0	0.000000
1	12347.0	23.681319
2	12348.0	57.975484
3	12349.0	24.076027
4	12350.0	19.670588
...	...	...
4367	18280.0	18.060000
4368	18281.0	11.545714
4369	18282.0	13.584615
4370	18283.0	2.771005
4371	18287.0	26.246857

4372 rows × 2 columns

In [28]:
```
#2) number of orders per customer

order_count = pd.DataFrame(df.groupby('CustomerID')['StockCode'].count()).reset_index()
order_count.rename(columns={'StockCode':'Order_Count'}, inplace=True)
order_count
```

Out[28]:

	CustomerID	Order_Count
0	12346.0	2
1	12347.0	182
2	12348.0	31
3	12349.0	73
4	12350.0	17
...	...	...
4367	18280.0	10
4368	18281.0	7
4369	18282.0	13
4370	18283.0	756
4371	18287.0	70

4372 rows × 2 columns

In [29]:
```
#3) customer country

countries = df[['CustomerID', 'Country']].drop_duplicates('CustomerID')
countries.head()
```

Out[29]:

	CustomerID	Country
0	17850.0	United Kingdom
9	13047.0	United Kingdom
26	12583.0	France
46	13748.0	United Kingdom
65	15100.0	United Kingdom

In [30]:
```
countries.isna().sum()
```

Out[30]:
```
CustomerID    1
Country       0
dtype: int64
```

In [31]:
```
countries.dropna(inplace=True)
countries.shape
```

Out[31]:
```
(4372, 2)
```

In [33]:
```
#4) most frequently purchased product
fav_product = df[['CustomerID', 'StockCode']]


# nice way to groupby customerID and the product they've ordered the most
fav_product = fav_product.groupby('CustomerID').agg(lambda x: x.value_counts().index[0])
fav_product
```

Out[33]:

	StockCode
CustomerID	
12346.0	23166
12347.0	22375
12348.0	POST
12349.0	23112
12350.0	21832
...	...
18280.0	22084
18281.0	23008
18282.0	23187
18283.0	21931
18287.0	85039B

4372 rows × 1 columns

In [34]:
```
#5) total spent by customer

order_val_sum = pd.DataFrame(df.groupby('CustomerID')['Order_Value'].sum()).reset_index()
order_val_sum.rename(columns={'Order_Value':'Total_Spent'}, inplace=True)
order_val_sum
```

Out[34]:

	CustomerID	Total_Spent
0	12346.0	0.00
1	12347.0	4310.00
2	12348.0	1797.24
3	12349.0	1757.55
4	12350.0	334.40
...	...	...
4367	18280.0	180.60
4368	18281.0	80.82
4369	18282.0	176.60
4370	18283.0	2094.88
4371	18287.0	1837.28

4372 rows × 2 columns

In [35]:
```
order_val_sum.shape
```

Out[35]: (4372, 2)

In [36]:
```
#6) most active month per customer (MOM)

new = df['InvoiceDate'].str.split('/', n=1, expand=True)
df['Month'] = new[0]
```

In [76]:
```
df.head(3)
```

Out[76]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	Order_Value	Month
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	12/1/2010 8:26	2.55	17850.0	United Kingdom	15.30	12
1	536365	71053	WHITE METAL LANTERN	6	12/1/2010 8:26	3.39	17850.0	United Kingdom	20.34	12
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	12/1/2010 8:26	2.75	17850.0	United Kingdom	22.00	12

In [77]:
```
# same agg method as above
MOM = df[['CustomerID', 'Month']].groupby('CustomerID').agg(lambda x: x.value_counts().index[0])
MOM
```

Out[77]:

	Month
CustomerID	
12346.0	1
12347.0	10
12348.0	12
12349.0	11
12350.0	2
...	...
18280.0	3
18281.0	6
18282.0	8
18283.0	11
18287.0	10

4372 rows × 1 columns

Merging tables

Now that I have the 6 tables representing information on unique customer level, I can re-join them back into one table by joining on the 'CustomerID' columns.

The six tables are:
1) **avg_ord_val** - average order value
2) **order_count** - number of orders
3) countries **- country**
4) fav_product **- favorite product**
5) order_val_sum **- total spent**
6) MOM** - most active month

In [39]:
```
customers_master = avg_ord_val.merge(order_count, on='CustomerID')
customers_master = customers_master.merge(countries, on='CustomerID')
customers_master = customers_master.merge(fav_product, on='CustomerID')
customers_master = customers_master.merge(order_val_sum, on='CustomerID')
customers_master = customers_master.merge(MOM, on='CustomerID')
```

In [40]:
```
customers_master.head()
```

Out[40]:

	CustomerID	Avg_Order_Value	Order_Count	Country	StockCode	Total_Spent	Month
0	12346.0	0.000000	2	United Kingdom	23166	0.00	1
1	12347.0	23.681319	182	Iceland	22375	4310.00	10
2	12348.0	57.975484	31	Finland	POST	1797.24	12
3	12349.0	24.076027	73	Italy	23112	1757.55	11
4	12350.0	19.670588	17	Norway	21832	334.40	2

In [41]:
```
customers_master.columns
```

Out[41]:
```
Index(['CustomerID', 'Avg_Order_Value', 'Order_Count', 'Country', 'StockCode',
       'Total_Spent', 'Month'],
      dtype='object')
```

In [42]:
```
# rename some columns
customers_master.rename(columns={'CustomerID':'customer_id','Avg_Order_Value':'avg_order_value',
                                 'Order_Count':'order_count','Country':'country','StockCode':'fav_prod
uct',
                                 'Total_Spent':'total_spent','Month':'most_active_month'}, inplace=Tru
e)

customers_master.head()
```

Out[42]:

	customer_id	avg_order_value	order_count	country	fav_product	total_spent	most_active_month
0	12346.0	0.000000	2	United Kingdom	23166	0.00	1
1	12347.0	23.681319	182	Iceland	22375	4310.00	10
2	12348.0	57.975484	31	Finland	POST	1797.24	12
3	12349.0	24.076027	73	Italy	23112	1757.55	11
4	12350.0	19.670588	17	Norway	21832	334.40	2

In [78]:
```
# save dataframe for modeling
#customers_master.to_pickle('customers_master.pkl')
```

In [79]:
```
#customers_master = pd.read_pickle('customers_master.pkl')
```

Converting categorical variables (country, fav_product) to numeric

In [45]:
```
customers_master.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4372 entries, 0 to 4371
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   customer_id        4372 non-null   float64
 1   avg_order_value    4372 non-null   float64
 2   order_count        4372 non-null   int64
 3   country            4372 non-null   category
 4   fav_product        4372 non-null   category
 5   total_spent        4372 non-null   float64
 6   most_active_month  4372 non-null   int64
 7   country_cat        4372 non-null   int8
 8   fav_product_cat    4372 non-null   int16
dtypes: category(2), float64(3), int16(1), int64(2), int8(1)
memory usage: 241.0 KB
```

In [46]:
```
# first change MOM to numeric

customers_master['most_active_month'] = pd.to_numeric(customers_master.most_active_month)
customers_master.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4372 entries, 0 to 4371
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   customer_id        4372 non-null   float64
 1   avg_order_value    4372 non-null   float64
 2   order_count        4372 non-null   int64
 3   country            4372 non-null   category
 4   fav_product        4372 non-null   category
 5   total_spent        4372 non-null   float64
 6   most_active_month  4372 non-null   int64
 7   country_cat        4372 non-null   int8
 8   fav_product_cat    4372 non-null   int16
dtypes: category(2), float64(3), int16(1), int64(2), int8(1)
memory usage: 241.0 KB
```

Using label encoding on rest of object variables

In [47]:
```
customers_master['country'] = customers_master['country'].astype('category')
customers_master.dtypes
```

Out[47]:
```
customer_id           float64
avg_order_value       float64
order_count             int64
country              category
fav_product          category
total_spent           float64
most_active_month       int64
country_cat              int8
fav_product_cat         int16
dtype: object
```

In [48]:
```
customers_master['country_cat'] = customers_master['country'].cat.codes
customers_master.head()
```

Out[48]:

	customer_id	avg_order_value	order_count	country	fav_product	total_spent	most_active_month	country_cat	fav_product_cat
0	12346.0	0.000000	2	United Kingdom	23166	0.00	1	35	862
1	12347.0	23.681319	182	Iceland	22375	4310.00	10	16	477
2	12348.0	57.975484	31	Finland	POST	1797.24	12	12	1296
3	12349.0	24.076027	73	Italy	POST	1757.55	11	18	1296
4	12350.0	19.670588	17	Norway	POST	334.40	2	24	1296

In [49]:
```
customers_master['fav_product'] = customers_master['fav_product'].astype('category')
customers_master.dtypes
```

Out[49]:
```
customer_id           float64
avg_order_value       float64
order_count             int64
country              category
fav_product          category
total_spent           float64
most_active_month       int64
country_cat              int8
fav_product_cat         int16
dtype: object
```

In [50]:
```
customers_master['fav_product_cat'] = customers_master['fav_product'].cat.codes
customers_master.head()
```

Out[50]:

	customer_id	avg_order_value	order_count	country	fav_product	total_spent	most_active_month	country_cat	fav_product_cat
0	12346.0	0.000000	2	United Kingdom	23166	0.00	1	35	862
1	12347.0	23.681319	182	Iceland	22375	4310.00	10	16	477
2	12348.0	57.975484	31	Finland	POST	1797.24	12	12	1296
3	12349.0	24.076027	73	Italy	POST	1757.55	11	18	1296
4	12350.0	19.670588	17	Norway	POST	334.40	2	24	1296

In [51]:
```
# save final dataframe, again
#customers_master.to_pickle('customers_master.pkl')
```

PCA

Conducting principle component analysis (PCA) to learn more about the importance of the variables and decide if they all are needed for modeling

In [52]:
```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

#for later
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
```

In [53]:
```
# isolate data to just the numerical columns (except CustomerID)
X = customers_master.iloc[:,1:].select_dtypes(exclude=['category'])
X.head()
```

Out[53]:

	avg_order_value	order_count	total_spent	most_active_month	country_cat	fav_product_cat
0	0.000000	2	0.00	1	35	862
1	23.681319	182	4310.00	10	16	477
2	57.975484	31	1797.24	12	12	1296
3	24.076027	73	1757.55	11	18	1296
4	19.670588	17	334.40	2	24	1296

In [54]:
```
# SCALE DATA (kmeans is distance based)

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

In [55]:
```
pca = PCA()
pca.fit(X_scaled)
```

Out[55]:
```
PCA(copy=True, iterated_power='auto', n_components=None, random_state=None,
    svd_solver='auto', tol=0.0, whiten=False)
```

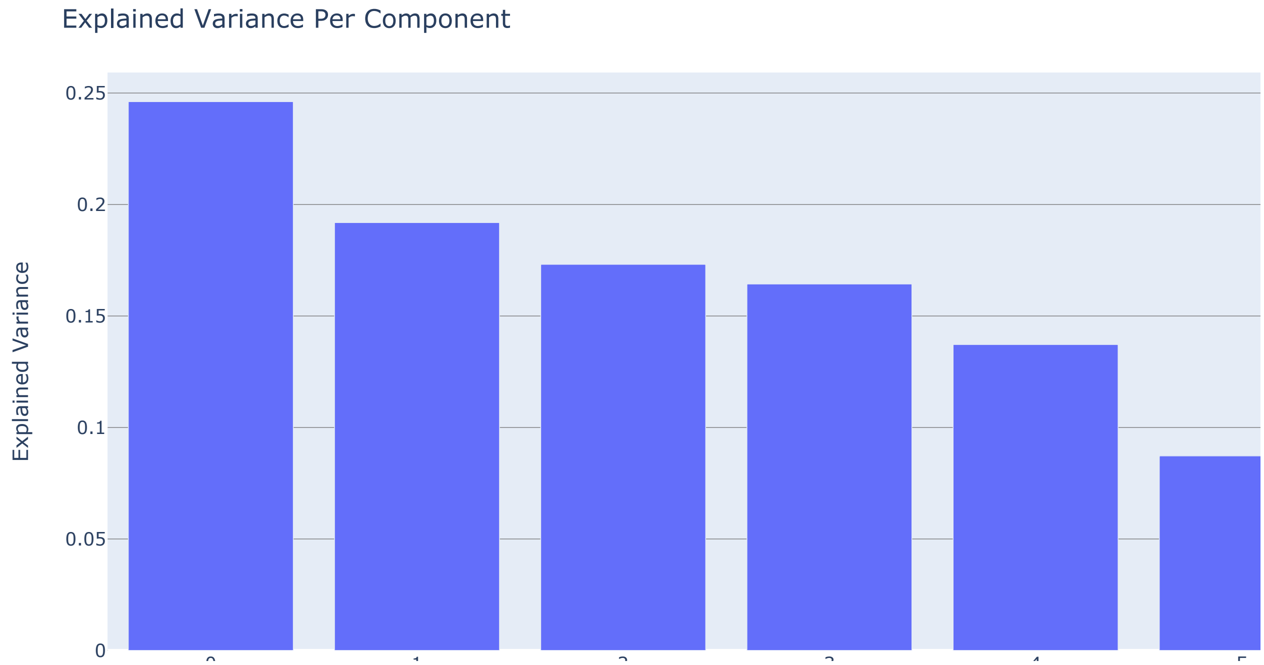
In [56]:
```
len(pca.components_)
```

Out[56]: 6

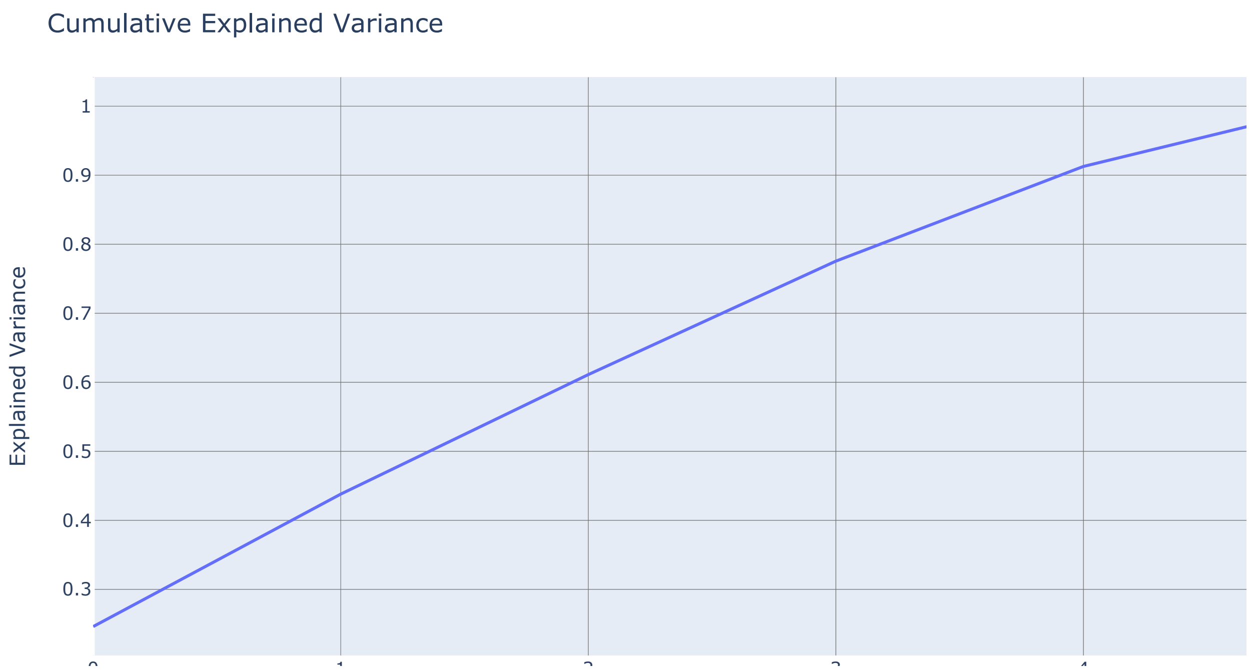
In [57]:
```
# Here we can see 24% of the total variance can be explained by just one PCA, 43% with 2, 61% with 3 a
nd so on...

print('Explained Variance Ratio =', sum(pca.explained_variance_ratio_[:3]))
```

```
Explained Variance Ratio = 0.611195786682605
```

In [58]:
```
# VIZ
fig = px.bar(y=pca.explained_variance_ratio_)

fig.update_layout(title='Explained Variance Per Component',
                  xaxis_title="Principle Components",
                  yaxis_title="Explained Variance")
```



In [59]:
```
# Cumulative variance explained

fig = px.line(y=np.cumsum(pca.explained_variance_ratio_))
fig.update_layout(title="Cumulative Explained Variance",
                  xaxis_title="Principle Components",
                  yaxis_title="Explained Variance")
```



The above shows we only need 5 components to explain +90% of the variance

In [60]:
```
# reduce data to just the 5 components

pca.n_components = 5
X_reduced = pca.fit_transform(X_scaled)
df_X_reduced = pd.DataFrame(X_reduced, index=X.index)
```

Finding Optimal Number of Clusters

In [61]:
```
# create helper function

def cluster(n_clusters):
    kmeans = KMeans(n_clusters=n_clusters)
    kmeans.fit(X_reduced)
    Z = kmeans.predict(X_reduced)
    return kmeans, Z
```

In [80]:
```
# Arbitrarily chosen, but large enough based on business problem understanding

max_clusters = 100
```

In [63]:
```
# Inertias represent the typical distance from a data point to its cluster's centroid.
# A.k.a. the mean squared distance between each instance and its closest centroid.

inertias = np.zeros(max_clusters)

for i in range(1, max_clusters):
    kmeans, Z = cluster(i)
    inertias[i] = kmeans.inertia_
```

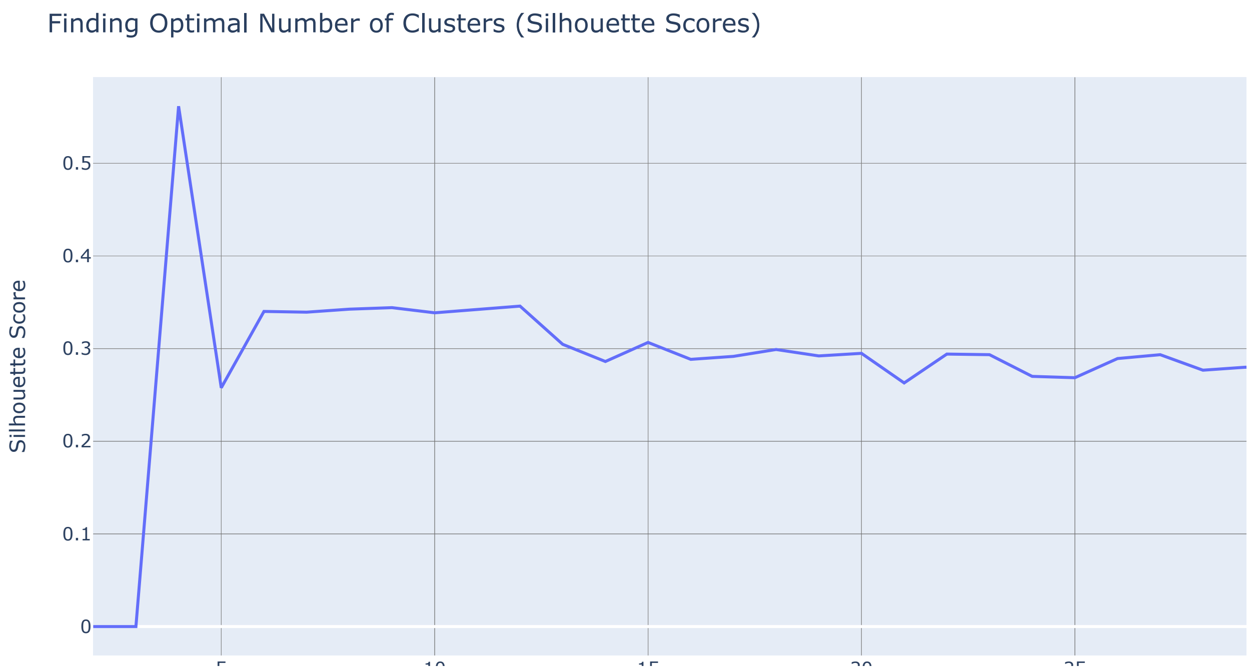
In [64]:
```
# Viz
fig = px.line(x=range(1,max_clusters+1),
             y=inertias)

fig.update_layout(
    title='Finding Optimal Number of Clusters (Elbow Method)',
    xaxis_title='Number of Clusters',
    yaxis_title = 'Intertia'
    )
```

The above (elbow) method is not easy to interpret since the curve is very smooth.
It looks as though the optimal number of clusters could be around 10.

Lets find the silhouette scores to get a better idea. (below)

In [81]:
```
# Silhouette score represents the distance between a point in one cluster from a point in another clus
ter.

reduced_max_clusters = 30

silhouette_scores = np.zeros(reduced_max_clusters)

for i in range(2, reduced_max_clusters):
    clusterer = KMeans(n_clusters=i)
    preds = clusterer.fit_predict(X_reduced)
    score = silhouette_score(X_reduced, preds)
    silhouette_scores[i] = score
```

In [82]:
```
#start from 2 here because 2 clusters was n_clusters starting point above
fig = px.line(x=range(2,reduced_max_clusters+2),
              y=silhouette_scores)

fig.update_layout(
    title='Finding Optimal Number of Clusters (Silhouette Scores)',
    xaxis_title='Number of Clusters',
    yaxis_title = 'Silhouette Score'
    )
```



Since the elbow method gave me a rough idea that the optimal number of clusters should be around 10, I then used 30 as the value for max_clusters when finding the silhoutte scores to determine a more precise optimal cluster value. The result of the silhoutte score method was 4. 4 also makes more sense for the customer segmentation problem I am trying to solve, so I decided to move foward with that as the proper cluster amount.

Final KMeans Modeling

In [68]:
```
n_clusters = 4
model, Z = cluster(n_clusters)
```

In [69]:
```
customers_master['segment'] = Z
```

In [70]:
```
customers_master.head()
```

Out[70]:

	customer_id	avg_order_value	order_count	country	fav_product	total_spent	most_active_month	country_cat	fav_product_cat	segm
0	12346.0	0.000000	2	United Kingdom	23166	0.00	1	35	862	
1	12347.0	23.681319	182	Iceland	22375	4310.00	10	16	477	
2	12348.0	57.975484	31	Finland	POST	1797.24	12	12	1296	
3	12349.0	24.076027	73	Italy	POST	1757.55	11	18	1296	
4	12350.0	19.670588	17	Norway	POST	334.40	2	24	1296	

In [71]:
```
# see how many customers are in each cluster
customers_master['segment'].value_counts()
```

Out[71]:
```
1    2173
0    1874
3     311
2      14
Name: segment, dtype: int64
```

In [92]:
```
# Visualizing final segments with first 2 components

fig = px.scatter(x=df_X_reduced[0],
           y=df_X_reduced[1],
          size = customers_master['order_count'],
           hover_name=customers_master['customer_id'],
          color=Z
          )

fig.update_layout(title = 'Final Customer Segments (clusters)', xaxis_title="Component 1",
                  yaxis_title="Component 2")
```