Importing Required Libraries

Online Retail Capstone Project

```
In [ ]: import pandas as pd
```

```
In [ ]: import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns
        import tensorflow as tf
        %matplotlib inline
        import datetime as dt
```

```
In [ ]: retail_file = r"C:\Users\hii\Desktop\data science\capstone project\retail final\ret
        retail = pd.read_excel(retail_file)
```

```
In [ ]: retail
```

```
In [ ]: retail.info()
```

```
In [ ]: retail.head()
```

```
In [ ]: retail.describe().T
```

```
In [ ]: retail = retail[retail['Quantity']>0]
```

```
In [ ]: retail.shape
```

```
In [ ]: retail = retail[['CustomerID','Quantity','UnitPrice','InvoiceDate','Country']]
```

finding of missing value

```
In [ ]: retail.isnull().sum()
```

```
In [ ]: retail.dtypes
```

```
In [ ]: retail = retail.dropna()
```

```
In [ ]: retail.shape
```

removal of duplicated data

```
In [ ]: retail.duplicated()
```

```
In [ ]: retail.duplicated().sum()
```

```
In [ ]: retail.drop_duplicates()
```

Performing some EDA to data

In [ ]:

```
plt.figure(figsize=(16,9))
retail.Country.value_counts().plot(kind='bar', title='Country-wise analysis of sal
plt.xlabel("Country")
plt.ylabel("Demand")
plt.show()
```

In [ ]:

```
print("The Minimum Unit Price: ", retail['UnitPrice'].min())
print("The Maximum Unit Price: ", retail['UnitPrice'].max())
print("The Number of Cutomers: ", len(np.unique(retail['CustomerID'].unique())))
print("Number of Regions : ",len(np.unique(retail['Country'].unique())))
print("Highest Quantity purchased : ",retail['Quantity'].max())
print("Lowest Quantity purchased : ",retail['Quantity'].min())
```

In [ ]:

```
retail.mean(axis=1)
```

In [ ]:

```
retail.median()
```

In [ ]:

```
retail.mode()
```

In [ ]:

```
retail.skew()
```

In [ ]:

```
retail.kurt()
```

creating cohort analysis

In [ ]:

```
def get_month(x):
    return dt.datetime(x.year, x.month, 1)
```

In [ ]:

```
retail['InvoiceMonth'] = retail['InvoiceDate'].apply(get_month)
```

In [ ]:

```
retail['InvoiceMonth']
```

In [ ]:

```
retail['CohortMonth'] =retail.groupby('CustomerID')['InvoiceMonth'].transform('min
```

In [ ]:

```
retail['CohortMonth']
```

In [ ]:

```
retail.head()
```

Calculating the time offset for each transaction allows you to evaluate the metrics for each cohort in a comparable fashion.def get_date(df, column):

In [ ]:

```
def get_date(dt1):
    year = dt1.dt.year
    month = dt1.dt.month
    day = dt1.dt.day
    return year,month,day
```

In [ ]:

```
invoice_year, invoice_month,_=get_date(retail['InvoiceMonth'])
```

```
In [ ]: invoice_month[:30]
```

```
In [ ]: cohort_year, cohort_month, _=get_date(retail['CohortMonth'])
```

```
In [ ]: cohort_month[:40]
```

Now calculate the differences of months from the invoice/transactional date and Cohort date for each customer

```
In [ ]: years_diff = invoice_year-cohort_year
        months_diff = invoice_month-cohort_month
        retail['CohortIndex'] = years_diff*12+months_diff+1
```

```
In [ ]: retail.head(10)
```

Counting number of unique customer Id's falling in each group of CohortMonth and CohortIndex

Counting daily active user from each cohort

```
In [ ]: cohort_data=retail.groupby(['CohortMonth', 'CohortIndex'])['CustomerID'].apply(pd.
```

```
In [ ]: cohort_counts=cohort_data.pivot_table(index='CohortMonth', columns='CohortIndex', 
```

```
In [ ]: cohort_data
```

```
In [ ]: cohort_counts
```

Now, we will calculate the retention count for each cohort Month paired with cohort Index

Now that we have a count of the retained customers for each cohortMonth and cohortIndex. We will calculate the retention rate

for each Cohort.

We will create a pivot table for this purpose.

```
In [ ]: cohort_size=cohort_counts.iloc[:,0]
        retention=cohort_counts.divide(cohort_size, axis=0)
```

In [ ]:

```
retention=retention.round(3)*100
```

In [ ]:

```
retention
```

The retention rate dataframe represents the value of CohortMonth accross CohortIndex. We can read it as follows:

On the cohort month 2011-03-01 the value ofretention for cohort index 6 is 16.8 That means 16.8% of customers were retained in the 6th month.

In [ ]:

```
retention.index = retention.index.strftime('%Y-%m')
```

In [ ]:

```
plt.figure(figsize=(16,9))
plt.title("Heatmap of the Retension rate", fontsize=14)
sns.heatmap(retention, annot = True,vmin = 0.0, vmax =20, cmap="YlGnBu", fmt='g')
plt.xlabel('Cohort Index')
plt.ylabel('Cohort Month')
plt.yticks( rotation='360')
plt.show()
```

In [ ]:

```
retail.head()
```

In [ ]:

```
import datetime as dt
```

For Recency, Calculate the number of days between present date and date of last purchase each customer.

For Frequency, Calculate the number of orders for each customer.

For Monetary, Calculate sum of purchase price for each customer.

In [ ]:

```
import datetime as dt
retail['TotalPrice'] = retail['UnitPrice']*retail['Quantity']
present =dt.datetime.now()
rfm = retail.groupby(['CustomerID']).aggregate({
    'InvoiceDate': lambda date: (present-date.max()).days,
    'Quantity': lambda freq: len(freq),
    'TotalPrice': lambda price: price.sum()
})
rfm.columns = ['Recency', 'Frequency', 'Monetory']
```

In [ ]:

```
rfm.head(1000)
```

```
In [ ]: rfm.dtypes
```

```
In [ ]: rfm.index = rfm.index.astype(int)
```

```
In [ ]: rfm.head()
```

```
In [ ]: rfm['r_ratings'] = pd.qcut(rfm['Recency'], 4, [1,2,3,4])
        rfm['f_ratings'] = pd.qcut(rfm['Frequency'], 4, [4,3,2,1])
        rfm['m_ratings'] = pd.qcut(rfm['Monetory'], 4, [4,3,2,1])
```

```
In [ ]: rfm.head(10)
```

Concat all the three ratings into one column

```
In [ ]: rfm['RFM_Score'] = rfm['r_ratings'].astype(str)+rfm['f_ratings'].astype(str)+rfm['
```

```
In [ ]: rfm.head()
```

Now Filter out the top 10 customers

```
In [ ]: rfm[rfm['RFM_Score']=='111'].sort_values(by='Monetory', ascending=False).head(10)
```

```
In [ ]: sns.set()
```

```
In [ ]: import warnings
        warnings.filterwarnings('ignore')
```

```
In [ ]: from sklearn.cluster import KMeans
        from mpl_toolkits.mplot3d import Axes3D
        from sklearn.preprocessing import MinMaxScaler
        import sklearn.metrics as sm
        from sklearn import datasets
        from sklearn.metrics import confusion_matrix, classification_report
```

```
In [ ]: retail.head()
```

```
In [ ]: retail=retail.drop(['CustomerID', 'Country', 'InvoiceDate', 'InvoiceMonth', 'Cohor
```

We have drop id column it seems that it does nat have any reference to form clusters

```
In [ ]: retail.head(500)
```

```
In [ ]: retail.isnull().sum()
```

Creating Clustering model KMeans Model

```
In [ ]: from sklearn.preprocessing import MinMaxScaler
```

```
mn=MinMaxScaler()
retail_sc=mn.fit_transform(retail)
```

In [ ]:
```
retail_sc_df=pd.DataFrame(retail_sc,columns=retail.columns,index=retail.index)
```

In [ ]:
```
retail_sc_df.head()
```

In [ ]:
```
from sklearn.cluster import KMeans
```

In [ ]:
```
km=KMeans(n_clusters=4)
```

In [ ]:
```
km.fit(retail_sc_df)
```

In [ ]:
```
km.labels_
```

In [ ]:
```
retail['cluster=4']=km.labels_
```

In [ ]:
```
retail.head(50000)
```

Evaluating Clustering Model

In [ ]:
```
km.inertia_
```

In [ ]:
```
from sklearn.metrics import silhouette_score
```

In [ ]:
```
silhouette_score(retail_sc_df,km.labels_)
```

Elbow Method to identify the Number Of Clusters

In [ ]:
```
wcss=[]

for i in range(1,15):
    km=KMeans(n_clusters=i, init="k-means++")
    km.fit(retail_sc_df)
    wcss.append(km.inertia_)

plt.figure(figsize=(10,5))
sns.lineplot(range(1,15), wcss, marker="o")
```

In [ ]:
```
print(km.labels_)
print(len(km.labels_))
```

In [ ]:
```
print(type(km.labels_))
unique,counts=np.unique(km.labels_,return_counts=True)
print(dict(zip(unique,counts)))
```

In [ ]:
```
retail['cluster']=km.labels_
sns.set_style('whitegrid')
sns.lmplot('UnitPrice','TotalPrice', data=retail,hue='cluster',palette='coolwarm',
```

In [ ]:
```
retail['cluster']=km.labels_
sns.set_style('whitegrid')
sns.lmplot('Quantity','TotalPrice', data=retail,hue='cluster',palette='coolwarm',s
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
```

```
In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:
```