RFM customer segmentation.ipynb

**Source Code:**

```
import numpy as np
import pandas as pd
import seaborn as sns
train = pd.read_excel('train.xlsx')
test = pd.read_excel('test.xlsx')
#append train test data
train['type']='train'
test['type']='test'
data = train.append(test, sort=False)
data.head()
data.info()
data.describe()
data.isnull().sum()
#remove null values
data.dropna(subset=['CustomerID'], inplace=True)
data.isnull().sum()

#Remove duplicate values
data[data.duplicated()].shape
data.drop_duplicates(inplace=True)
#Reset the index
indexcol=np.array(list(range(0,len(data))))
data.set_index(indexcol,inplace=True)
#Remove the orders that were reversed
data = data[data['Quantity'] > 0]
data.shape
data.head()
# Transfer the dataframe into a file for analysis in Tableau
#data.to_csv('file.csv')
#Customers who ordered more than once
n_orders = data.groupby(['CustomerID'])['InvoiceNo'].nunique()
mult_orders_perc = np.sum(n_orders > 1) / data['CustomerID'].nunique()
print(f'{100 * mult_orders_perc:.2f}% of customers ordered more than once.')
import datetime as dt
#Perform cohort analysis
def get_month(x) : return dt.datetime(x.year,x.month,1)
data['InvoiceMonth'] = data['InvoiceDate'].apply(get_month)
grouping = data.groupby('CustomerID')['InvoiceMonth']
data['CohortMonth'] = grouping.transform('min')
data.tail()
def get_month_int (dframe,column):
```

```
    year = dframe[column].dt.year
    month = dframe[column].dt.month
    day = dframe[column].dt.day
    return year, month , day

invoice_year,invoice_month,_ = get_month_int(data,'InvoiceMonth')
cohort_year,cohort_month,_ = get_month_int(data,'CohortMonth')

year_diff = invoice_year - cohort_year
month_diff = invoice_month - cohort_month

data['CohortIndex'] = year_diff * 12 + month_diff + 1
#Count monthly active customers from each cohort
grouping = data.groupby(['CohortMonth', 'CohortIndex'])
cohort_data = grouping['CustomerID'].apply(pd.Series.nunique)
# Return number of unique elements in the object.
cohort_data = cohort_data.reset_index()
cohort_counts = cohort_data.pivot(index='CohortMonth',columns='CohortIndex',values='CustomerID')
cohort_counts
# Retention table
cohort_size = cohort_counts.iloc[:,0]
retention = cohort_counts.divide(cohort_size,axis=0) #axis=0 to ensure the divide along the row axis
retention.round(3) * 100 #to show the number as percentage
#Build the heatmap
import matplotlib.pyplot as plt
plt.figure(figsize=(15, 8))
plt.title('Retention rates')
sns.heatmap(data=retention,annot = True,fmt = '.0%',vmin = 0.0,vmax = 0.5,cmap="BuPu_r")
plt.show()
#Average quantity for each cohort
grouping = data.groupby(['CohortMonth', 'CohortIndex'])
cohort_data = grouping['Quantity'].mean()
cohort_data = cohort_data.reset_index()
average_quantity = cohort_data.pivot(index='CohortMonth',columns='CohortIndex',values='Quantity')
average_quantity.round(1)
average_quantity.index = average_quantity.index.date

#Build the heatmap
plt.figure(figsize=(15, 8))
plt.title('Average quantity for each cohort')
sns.heatmap(data=average_quantity,annot = True,vmin = 0.0,vmax =20,cmap="BuGn_r")
plt.show()
```

# RFM

```
#Recency
data['Recent']=(pd.to_datetime(data['InvoiceDate'].max()) - pd.to_datetime(data['InvoiceDate'])).dt.days
data = data[data['Recent'] <= 366]
```

```
Recency=data.groupby(['CustomerID'], as_index=False)['Recent'].max()
Recency.columns = ['CustomerID','Recency']
Recency.head()
Recency.shape
#Frequency
Frequency = data.groupby(['CustomerID'], as_index=False)['InvoiceNo'].count()
Frequency.columns = ['CustomerID','Frequency']
Frequency.head()
#Monetary
data['Totalsales'] = data['Quantity']*data['UnitPrice']
Monetary = data.groupby(['CustomerID'], as_index=False)['Totalsales'].agg('sum')
Monetary.columns = ['CustomerID','Monetary']
Monetary.head()
temp_df = pd.merge(Recency,Frequency,on='CustomerID')
RFM_metrics = pd.merge(temp_df,Monetary,on='CustomerID')
RFM_metrics.head()
#RFM_metrics.to_csv('file_1.csv')
quantiles = RFM_metrics.quantile(q=[0.25,0.5,0.75])

quantiles
quantiles.to_dict()
#Define function for the most frequent and high spending customer
def FMscore(x,c,d):
    if x <= d[c][0.25]:
        return 1
    elif x <= d[c][0.50]:
        return 2
    elif x <= d[c][0.75]:
        return 3
    else:
        return 4
#Define function for the most Recent customer
def Rscore(x,c,d):
    if x <= d[c][0.25]:
        return 4
    elif x <= d[c][0.50]:
        return 3
    elif x <= d[c][0.75]:
        return 2
    else:
        return 1
rfm_segmentation = RFM_metrics
rfm_segmentation['R_Quartile'] = rfm_segmentation['Recency'].apply(Rscore, args=('Recency',quantiles))
rfm_segmentation['F_Quartile'] = rfm_segmentation['Frequency'].apply(FMscore, args=('Frequency',quantiles))
rfm_segmentation['M_Quartile'] = rfm_segmentation['Monetary'].apply(FMscore, args=('Monetary',quantiles))
rfm_segmentation.head()
#RFM segment
```

```
rfm_segmentation['RFMSegment'] = rfm_segmentation['R_Quartile'].map(str) + rfm_segmentation['F_Quartile']
.map(str) + rfm_segmentation['M_Quartile'].map(str)
rfm_segmentation['RFMScore'] = rfm_segmentation.R_Quartile + rfm_segmentation.F_Quartile + rfm_segment
ation.M_Quartile
rfm_segmentation.head()
rfm_segmentation = rfm_segmentation.sort_values('RFMScore',ascending=False)
#rfm_segmentation.to_csv('rfm.csv')
rfm_segmentation.head()
print("Best Customers: {}".format(len(rfm_segmentation[rfm_segmentation['RFMScore'] == 12])))
print("Frequent Customers: {}".format(len(rfm_segmentation[rfm_segmentation['F_Quartile'] == 4])))
print("Money spending Customers: {}".format(len(rfm_segmentation[rfm_segmentation['M_Quartile'] == 4])))
print("Lost Customers: {}".format(len(rfm_segmentation[rfm_segmentation['RFMScore'] == 3])))
#Normalize and standardize the data
rfm_segmentation.skew()
#UnitPrice
sns.boxplot(rfm_segmentation['Frequency'])
#Quantity
sns.boxplot(rfm_segmentation['Monetary'])
#Remove Outliers
rfm_segmentation1 = rfm_segmentation[(rfm_segmentation.Frequency < 3000 ) & (rfm_segmentation.Monetar
y < 10000)]
rfm_segmentation1.skew()
rfm_segmentation1['Frequency'] = np.log(rfm_segmentation1['Frequency']+0.01)
rfm_segmentation1['Monetary'] = np.log(rfm_segmentation1['Monetary']+0.01)
rfm_segmentation1.skew()
rfm_segmentation1.shape
sns.pairplot(rfm_segmentation1,diag_kind='kde');
final = rfm_segmentation1.iloc[:,1:4]
#Standardise the data

from sklearn.preprocessing import StandardScaler
scale = StandardScaler()
data2 = scale.fit_transform(final)
data2
final.head()
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples, silhouette_score

import matplotlib.pyplot as plt
import matplotlib.cm as cm
import numpy as np

print(__doc__)

X = data2

range_n_clusters = [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
```

```
for n_clusters in range_n_clusters:
    # Create a subplot with 1 row and 2 columns
    fig, (ax1, ax2) = plt.subplots(1, 2)
    fig.set_size_inches(18, 7)

    # The 1st subplot is the silhouette plot
    # The silhouette coefficient can range from -1, 1 but in this example all
    # lie within [-0.1, 1]
    ax1.set_xlim([-0.1, 1])
    # The (n_clusters+1)*10 is for inserting blank space between silhouette
    # plots of individual clusters, to demarcate them clearly.
    ax1.set_ylim([0, len(X) + (n_clusters + 1) * 10])

    # Initialize the clusterer with n_clusters value and a random generator
    # seed of 10 for reproducibility.
    clusterer = KMeans(n_clusters=n_clusters, random_state=10)
    cluster_labels = clusterer.fit_predict(X)

    # The silhouette_score gives the average value for all the samples.
    # This gives a perspective into the density and separation of the formed
    # clusters
    silhouette_avg = silhouette_score(X, cluster_labels)
    print("For n_clusters =", n_clusters,
          "The average silhouette_score is :", silhouette_avg)

    # Compute the silhouette scores for each sample
    sample_silhouette_values = silhouette_samples(X, cluster_labels)

    y_lower = 10
    for i in range(n_clusters):
        # Aggregate the silhouette scores for samples belonging to
        # cluster i, and sort them
        ith_cluster_silhouette_values = \
            sample_silhouette_values[cluster_labels == i]

        ith_cluster_silhouette_values.sort()

        size_cluster_i = ith_cluster_silhouette_values.shape[0]
        y_upper = y_lower + size_cluster_i

        color = cm.nipy_spectral(float(i) / n_clusters)
        ax1.fill_betweenx(np.arange(y_lower, y_upper),
                          0, ith_cluster_silhouette_values,
                          facecolor=color, edgecolor=color, alpha=0.7)

        # Label the silhouette plots with their cluster numbers at the middle
```

```
        ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))

        # Compute the new y_lower for next plot
        y_lower = y_upper + 10  # 10 for the 0 samples

    ax1.set_title("The silhouette plot for the various clusters.")
    ax1.set_xlabel("The silhouette coefficient values")
    ax1.set_ylabel("Cluster label")

    # The vertical line for average silhouette score of all the values
    ax1.axvline(x=silhouette_avg, color="red", linestyle="--")

    ax1.set_yticks([])  # Clear the yaxis labels / ticks
    ax1.set_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])

    # 2nd Plot showing the actual clusters formed
    colors = cm.nipy_spectral(cluster_labels.astype(float) / n_clusters)
    ax2.scatter(X[:, 0], X[:, 1], marker='.', s=30, lw=0, alpha=0.7,
                c=colors, edgecolor='k')

    # Labeling the clusters
    centers = clusterer.cluster_centers_
    # Draw white circles at cluster centers
    ax2.scatter(centers[:, 0], centers[:, 1], marker='o',
                c="white", alpha=1, s=200, edgecolor='k')

    for i, c in enumerate(centers):
        ax2.scatter(c[0], c[1], marker='$%d$' % i, alpha=1,
                    s=50, edgecolor='k')

    ax2.set_title("The visualization of the clustered data.")
    ax2.set_xlabel("Feature space for the 1st feature")
    ax2.set_ylabel("Feature space for the 2nd feature")

    plt.suptitle(("Silhouette analysis for KMeans clustering on sample data "
                  "with n_clusters = %d" % n_clusters),
                 fontsize=14, fontweight='bold')

plt.show()
#Since the silhouette score is more at clusters 2, we will use the value
model = KMeans(n_clusters=2)
model.fit(data2)
group = model.predict(data2)
final.head()
final['group'] = group
# to plot in tableau
#final.to_csv('cluster_data.csv')
```

```
sns.pairplot(final,diag_kind='kde',hue='group');
#Elbow method to plot in tableau
wcss=[]
cluster=[]
for i in range(1,10):
    model = KMeans(n_clusters=i)
    model.fit(final)
    wcss.append(model.inertia_)
    cluster.append(i)
from pandas import DataFrame
plot = list(zip(cluster,wcss))
df = DataFrame(plot,columns=['No of cluster','error'])
#df.to_csv('error.csv')
```