Online Retail Data Analysis

**Introduction:**

The purpose of this exercise is to analyze online retail data collected between December 1st, 2010 and December 9th, 2011. The data are extracted from an excel file that contains purchases from an online retail store.

**Data preparation and cleaning:**

Received unorganized text file that kept track of all purchases (or returns). Input variables consisted of InvoiceNo, StockCode, Description, Quantity, InvoiceDate, UnitPrice, CustomerID, and Country. Head of data depicted below:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **InvoiceNo** | **StockCode** | **Description** | **Quantity** | **InvoiceDate** | **UnitPrice** | **CustomerID** | **Country** |
| **536365** | 85123A | WHITE HANGING HEART T-LIGHT HOLDER | 6 | 12/1/2010 8:26 | 2.55 | 17850 | United Kingdom |
| **536365** | 71053 | WHITE METAL LANTERN | 6 | 12/1/2010 8:26 | 3.39 | 17850 | United Kingdom |
| **536365** | 84406B | CREAM CUPID HEARTS COAT HANGER | 8 | 12/1/2010 8:26 | 2.75 | 17850 | United Kingdom |
| **536365** | 84029G | KNITTED UNION FLAG HOT WATER BOTTLE | 6 | 12/1/2010 8:26 | 3.39 | 17850 | United Kingdom |
| **536365** | 84029E | RED WOOLLY HOTTIE WHITE HEART. | 6 | 12/1/2010 8:26 | 3.39 | 17850 | United Kingdom |
| **536365** | 22752 | SET 7 BABUSHKA NESTING BOXES | 2 | 12/1/2010 8:26 | 7.65 | 17850 | United Kingdom |
| **536365** | 21730 | GLASS STAR FROSTED T-LIGHT HOLDER | 6 | 12/1/2010 8:26 | 4.25 | 17850 | United Kingdom |
| **536366** | 22633 | HAND WARMER UNION JACK | 6 | 12/1/2010 8:28 | 1.85 | 17850 | United Kingdom |
| **536366** | 22632 | HAND WARMER RED POLKA DOT | 6 | 12/1/2010 8:28 | 1.85 | 17850 | United Kingdom |
| **536367** | 84879 | ASSORTED COLOUR BIRD ORNAMENT | 32 | 12/1/2010 8:34 | 1.69 | 13047 | United Kingdom |
| **536367** | 22745 | POPPY'S PLAYHOUSE BEDROOM | 6 | 12/1/2010 8:34 | 2.1 | 13047 | United Kingdom |
| **536367** | 22748 | POPPY'S PLAYHOUSE KITCHEN | 6 | 12/1/2010 8:34 | 2.1 | 13047 | United Kingdom |

As you can see, there is a unique row for the invoice number and stock code. This was particularly messy, because I needed to form the table in a way where each row would contain a single customer ID. Handling this data set and preforming manipulations on it was a crucial part of the project, as I needed relevant variables for each unique customer.

To begin, I used listwise deletion on missing values. Generally speaking, listwise deletion is not good on smaller datasets because it eliminates a good amount of the data. In this case, the rows with missing values (3,710 of 541,909, or 0.685%) were removed from the original dataset. Therefore, it was assumed that listwise deletion on this large of a dataset had an insignificant effect on the results of analysis of the data.

Feature engineering was a much more time-consuming and tedious task. In order to properly access the invoice date, I would have to translate it into a datetime object, which was much easier to work with.

# Makes date more readable and program friendly  
data['InvoiceDate'] = pd.to\_datetime(data.InvoiceDate).dt.date

This large step was the most important part of engineering the features, as I would create the features that I would use for the rest of the analysis. The total revenue for each customer would consist of multiplying the quantity and unit price of each row, and then adding up all of them for each customer. Feature engineering also enabled me to find the maximum and minimum date of purchase for each customer. The last step involved grouping by unique customer ID. My end result was a newly ordered dataset with each unique customer ID having their own row, with the extracted features.

# Creating new columns to apply functions on, needed for grouping step  
data['Total Revenue'] = data['Quantity'] \* data['UnitPrice']  
data['Number Of Purchases'] = data['Quantity']  
data['Days From Last Purchase'] = data['InvoiceDate']  
data['Days From First Purchase'] = data['InvoiceDate']  
  
# Find most recently recorded date in dataframe  
# This will be needed so that we can subtract a given date from it  
column = data["Days From Last Purchase"]  
max\_value = column.max()  
  
# Subtract each each date recorded from the most recent date recorded, to find number of days elapsed since then  
data['Days From Last Purchase'] = max\_value - data['Days From Last Purchase']  
data['Days From First Purchase'] = max\_value - data['Days From First Purchase']  
  
# For each customer, we get their unique statistics such as total revenue spent, total number of purchases,  
# most date of most recent purchase, and date of earliest purchase  
# This code makes a unique table for each customer id, finds the sum of all of their purchases, min days  
# from last purchase, max days from first purchase, and sum of all the revenue.  
# Thus, for each customer, their own unique row is created containing the features listed below.  
data = data.groupby(['CustomerID']).agg({'Number Of Purchases': ['sum'], 'Days From Last Purchase': ['min'],  
 'Days From First Purchase': ['max'], 'Total Revenue': ['sum']})

Thus, I was able to now work with this prepared dataset:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **CustomerID** | **Number Of Purchases** | **Days From Last Purchase** | **Days From First Purchase** | **Total Revenue** |
| **0** | 15573.0 | 363 | 31.0 | 85.0 | 607 |
| **1** | 14711.0 | 966 | 10.0 | 262.0 | 2838 |
| **2** | 17763.0 | 12 | 263.0 | 263.0 | 15 |
| **3** | 16987.0 | 402 | 3.0 | 205.0 | 1625 |
| **4** | 16365.0 | 504 | 17.0 | 367.0 | 1540 |
| **5** | 17720.0 | 576 | 26.0 | 367.0 | 1195 |
| **6** | 18225.0 | 3206 | 3.0 | 371.0 | 5361 |
| **7** | 13340.0 | 5770 | 45.0 | 276.0 | 10736 |
| **8** | 13884.0 | 297 | 7.0 | 283.0 | 781 |
| **9** | 17861.0 | 1968 | 3.0 | 319.0 | 2059 |
| **10** | 15157.0 | 2242 | 3.0 | 358.0 | 1869 |
| **12** | 16152.0 | 642 | 268.0 | 268.0 | 1829 |

**K-means clustering:**

The end result of this project was to find different clusters for the 4 variables and explore their differences.

A pipeline was created to scale the data where all 4 variables had the same units. I chose MinMaxScalar as I did not want outliers to dictate the clusters. Testing out RobustScalar, most data points were assigned a single cluster while the largest outliers shared the other 3. With MinMax, all 4 clusters had a balanced number of customers in them.

In the same pipeline, I was able to apply the kmeans method. This led me to pickle the model, as kmeans would not have to calculate every time I ran the program.

Soon after, I was able to predict the clusters for each customer in my dataset:

# Find which cluster each customer belongs to in out dataset  
# Again, not including the customer id column because its numerical value has no say in the customers habits  
# Creates new column for each customer noting the cluster they belong to.  
# We now have our dataset where each customer has a column for the cluster they belong to  
data['Cluster Category'] = pd.Series(pipelineClustering.predict(data[['Number Of Purchases',  
 'Days From Last Purchase',  
 'Days From First Purchase',  
 'Total Revenue']].\_get\_numeric\_data().dropna(axis=1)),  
 index=data.index)

**Each of my 4 clusters defined:**

# Clusters are: Highest spenders/most loyal customers, new customers, infrequent customers,  
# and former customers/lowest spenders.

**Predicting cluster from user input:**

The program prompts the user to enter in their customer features and the cluster they belong to depending on their features is returned.

**Supervised machine learning:**

As a bonus, I played around and used the dataset I created including the clusters for each customer and decided to make a model to train on that.

print('Supervised learning model:')  
X\_train, X\_test, y\_train, y\_test = \  
 train\_test\_split(data[['Number Of Purchases',  
 'Days From Last Purchase',  
 'Days From First Purchase',  
 'Total Revenue']], data['Cluster Category'], test\_size=0.3, random\_state=0)  
# Pipeline  
pipeline\_randomforest = Pipeline([('scalar', MinMaxScaler()), ('rf\_classifier', RandomForestClassifier())])  
pipeline\_randomforest.fit(X\_train, y\_train)  
# The random forest model had the best accuracy out of any other models I tested previously  
# Accuracy is between 98-99%  
print('Testing accuracy: ')  
print(pipeline\_randomforest.score(X\_test, y\_test))