**CIS-660 Lab 1**

**Part 3 and Part 4**

**Submitted By:**

A person standing in front of a bridge

Description automatically generated with medium confidence

**Name: Mohamed Gani Mohamed Sulthan**

**CSU ID: 2811619**

**Grail Id: momohame@grail.eecs.csuohio.edu**

**Master’s in computer science**

**Part 3: Calculating Proximity of Two Binary Object Vectors With Simple Matching, Jaccard Similarity, Cosine Similarity**

Make sure each attribute is transformed in a same scale for numeric attributes and Binarization for each nominal attribute, and each discretized numeric attribute to standardization.

Make sure to apply a correct similarity measure for nominal (one hot encoding)/binary attributes and numeric attributes respectively. You can bring indicator variable or a different weight for each attribute as discussed in class.

1. **Calculate Similarity in Simple Matching, Extended Jaccard Similarity, and Cosine Similarity between two following objects of your transformed input data**

**- CustomerKey: 11000 and CustomerKey: 11001**

**Step 1:** To calculate the similarities we need to fetch the respective rows using dataFrame.loc function

**Step 2:** After fetching the two values from the dataframe started doing the simple matching, extended jaccard similarity and cosine similarity.

**Step 3:** For that we need to remove the CustomerKey and GeographyKey to predict accurately

**Step 4**: I have written three methods, simple\_matching, jaccard\_similarity and cosine\_similarity

**Code**:

#Calculating similarities

#Fetch the required rows

calculating\_similarity = dataFrame.loc[dataFrame['CustomerKey'].isin([11000, 11001])]

print(calculating\_similarity)

#Dropping the CustomerKey and GeographyKey to get accurate values

calculating\_similarity = calculating\_similarity.drop(columns=['CustomerKey', 'GeographyKey'])

calculating\_similarity = calculating\_similarity.values.tolist()

print(calculating\_similarity)

#Assiging list in to two lists

length = len(calculating\_similarity)

# To seprate the list

middle\_index = length//length

custKey1 = calculating\_similarity[:middle\_index]

custKey2 = calculating\_similarity[middle\_index:]

print(custKey1)

print(custKey2)

Text

Description automatically generated

Graphical user interface, text, application

Description automatically generated

1. **Calculate Similarity in Simple Matching, Extended Jaccard Similarity, and Cosine Similarity between two following objects of your transformed input data**

**- CustomerKey: 11000 and CustomerKey: 11012**

**Step 1:** To calculate the similarities we need to fetch the respective rows using dataFrame.loc function

**Step 2:** After fetching the two values from the dataframe started doing the simple matching, extended jaccard similarity and cosine similarity.

**Step 3:** For that we need to remove the CustomerKey and GeographyKey to predict accurately

**Step 4**: I have written three methods, simple\_matching, jaccard\_similarity and cosine\_similarity

**Code**:

#Calculating similarities

#Fetch the required rows

calculating\_similarity = dataFrame.loc[dataFrame['CustomerKey'].isin([11000, 11012])]

print(calculating\_similarity)

#Dropping the CustomerKey and GeographyKey to get accurate values

calculating\_similarity = calculating\_similarity.drop(columns=['CustomerKey', 'GeographyKey'])

calculating\_similarity = calculating\_similarity.values.tolist()

print(calculating\_similarity)

#Assiging list in to two lists

length = len(calculating\_similarity)

# To seprate the list

middle\_index = length//length

custKey1 = calculating\_similarity[:middle\_index]

custKey2 = calculating\_similarity[middle\_index:]

print(custKey1)

print(custKey2)

Text

Description automatically generated

Graphical user interface, text, application

Description automatically generated

**Part 4: Correlation Analysis by Building a Correlation Matrix**

1. **Build a Correlation Matrix for Every Pair of the Features for the entire Data Set**
2. **Visualize Your Correlation Matrix of All the Features in a Heat Map.**

Step 1: Using the dataFrame function corr() to get the correlation matrix for

the data frame

Step 2: Display the correlation matrix using matplotlib

#Building Correlartion Matrix for dataframe

dataFramecorrMatrix = dataFrame.corr()

print (dataFramecorrMatrix)

plt.figure(figsize=(18, 12), dpi=80)

sn.heatmap(dataFramecorrMatrix, annot=True)

Graphical user interface, application

Description automatically generated

Graphical user interface, application

Description automatically generated

Graphical user interface, chart

Description automatically generated

Chart

Description automatically generated

1. **Divide your data set into two data sets: One with Bike Buyer = 1 and the other set with Bike Buyer = 0**

#Dividing dataframe by BikeBuyer = 0 and BikeBuyer = 1

bikeBuyer\_0 = dataFrame[dataFrame['BikeBuyer'] == 0]

bikeBuyer\_1 = dataFrame[dataFrame['BikeBuyer'] == 1]

**Dataset with BikeBuyer = 0**

A screenshot of a computer

Description automatically generated

**Dataset with BikeBuyer = 1**

A screenshot of a computer

Description automatically generated

1. **Then build a Correlation Matrix for Every pair of the Features for the record set with Bike Buyer = 1 and the other record set with Bike Buyer = 0 respectively.**

**BikeBuyer\_0 Correlation Matrix:**

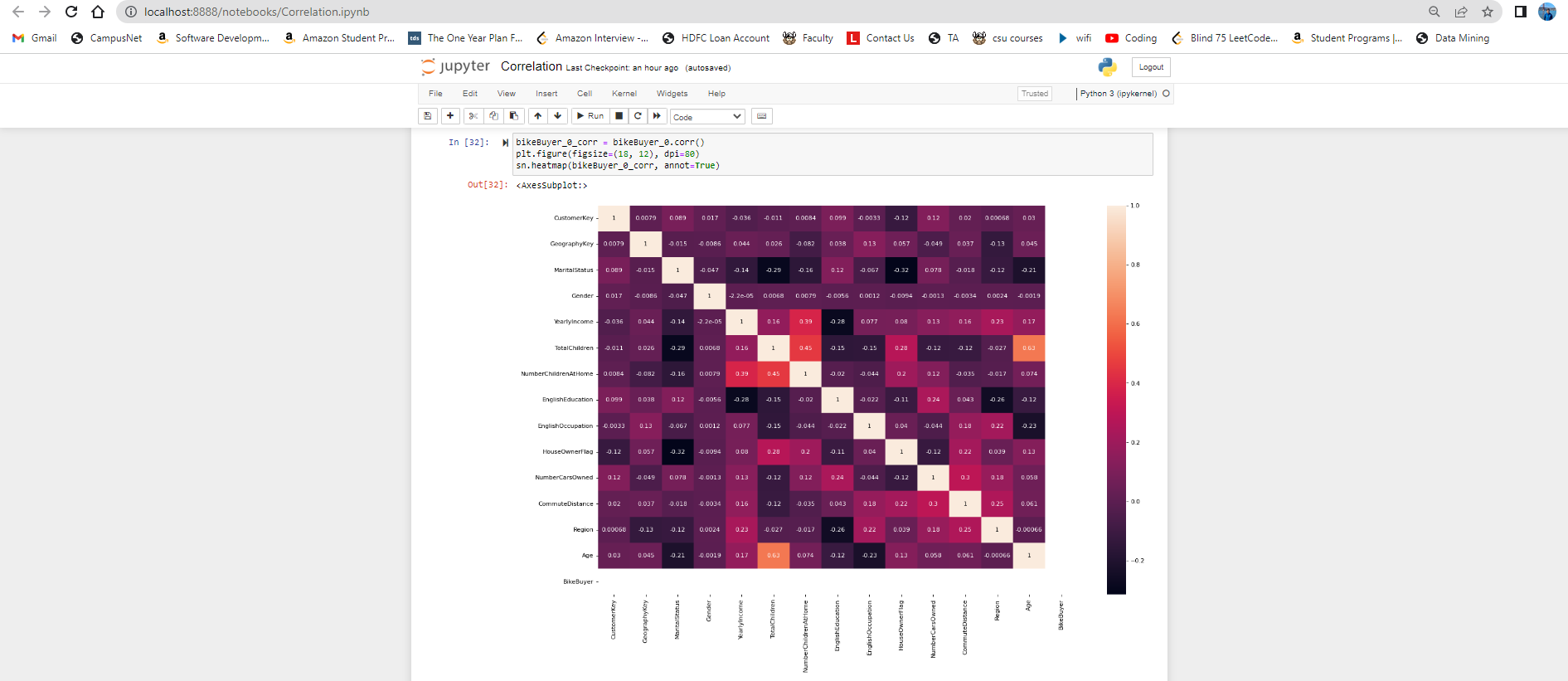
**Code:**

#Building Correlartion Matrix for bikeBuyer0

bikeBuyer\_0\_corr = bikeBuyer\_0.corr()

plt.figure(figsize=(18, 12), dpi=80)

sn.heatmap(bikeBuyer\_0\_corr, annot=True)



**BikeBuyer\_1 Correlation Matrix:**

**Code:**

#Building Correlartion Matrix for bikeBuyer1

bikeBuyer\_1\_corr = bikeBuyer\_1.corr()

plt.figure(figsize=(18, 12), dpi=80)

sn.heatmap(bikeBuyer\_1\_corr, annot=True)

Graphical user interface, chart

Description automatically generated

1. **From three correlation matrix built from 1 and 3 above, Compare the Correlation values between two features Age and Yearly Income with the Correlation between two features Commute Distance and Yearly Income.**

To do the correlation matrix for the selected features, get the specific columns from the dataframe and do the corr function with the selected another column from the dataframe

**Code:**

#Comparing two features(Age and YearlyIncome) from alll the selected dataframes

print('Full Correlation for Age and YearlyIncome', dataFrame['Age'].corr(dataFrame['YearlyIncome']))

print('Bike buyer 0 Correlation for Age and YearlyIncome', bikeBuyer\_0\_corr['Age'].corr(bikeBuyer\_0\_corr['YearlyIncome']))

print('Bike buyer 1 Correlation for Age and YearlyIncome', bikeBuyer\_1\_corr['Age']. corr(bikeBuyer\_1\_corr['YearlyIncome']))

Application

Description automatically generated with medium confidence

A picture containing application

Description automatically generated

1. **Compare and discuss which two features are correlated more strongly than the others for each data set from 1 and 3.**

Step 1: I have taken all the three dataframes to do the strongly correlation with the other attributes.

Step 2: By using upper function we can get the strongly correlation difference in the dataframe

Step 3: We can set the correlation value and filter it accordingly

**Code:**

#Create correlation matrix #Select upper triangle of correlation matrix #Find features with correlation greater than 0.6

strong\_corr\_matrix\_df = dataFrame.corr().abs()

upper = strong\_corr\_matrix\_df.where(np.triu(np.ones(strong\_corr\_matrix\_df.shape), k=1).astype(np.bool))

strongly\_corr\_df = [column for column in upper.columns if any(upper[column] > 0.5)]

print("Strongly Correlated for actual DF", strongly\_corr\_df)

strong\_corr\_matrix\_b0 = bikeBuyer\_0\_corr.corr().abs()

upper = strong\_corr\_matrix\_b0.where(np.triu(np.ones(strong\_corr\_matrix\_b0.shape), k=1).astype(np.bool))

strongly\_corr\_b0 = [column for column in upper.columns if any(upper[column] > 0.6)]

print("Strongly Correlated for BikeBuyer 0 DF", strongly\_corr\_b0)

strong\_corr\_matrix\_b1 = bikeBuyer\_1\_corr.corr().abs()

upper = strong\_corr\_matrix\_b1.where(np.triu(np.ones(strong\_corr\_matrix\_b1.shape), k=1).astype(np.bool))

strongly\_corr\_b1 = [column for column in upper.columns if any(upper[column] > 0.6)]

print("Strongly Correlated for BikeBuyer 1 DF", strongly\_corr\_b1)

A picture containing text

Description automatically generated

**Strongly Correlated two features:**

Step 1: From the selected attributes, I did the plot using matplotlib.pyplot module

Step 2: By assigning the x- axis and y-axis from the strongly correlated selected attributes, I plotted the graph

Step 3: Below are the different highly correlated features for three different dataframes

**Code:**

print('Scatter plotting for actual dataframe')

plt.scatter(dataFrame['TotalChildren'],dataFrame['Age'])

plt.plot(np.unique(dataFrame['TotalChildren']), np.poly1d(np.polyfit(dataFrame['TotalChildren'], dataFrame['Age'], 1)) (np.unique(dataFrame['TotalChildren'])),color='red')

print('Scatter plotting for bike buyer 0 dataframe')

plt.scatter(bikeBuyer\_0['TotalChildren'],bikeBuyer\_0['NumberChildrenAtHome'])

plt.plot(np.unique(bikeBuyer\_0['TotalChildren']), np.poly1d(np.polyfit(bikeBuyer\_0['TotalChildren'], bikeBuyer\_0['NumberChildrenAtHome'], 1)) (np.unique(bikeBuyer\_0['TotalChildren'])),color='red')

print('Scatter plotting for bike buyer 1 dataframe')

plt.scatter(bikeBuyer\_1['HouseOwnerFlag'],bikeBuyer\_1['Age'])

plt.plot(np.unique(bikeBuyer\_1['HouseOwnerFlag']), np.poly1d(np.polyfit(bikeBuyer\_1['HouseOwnerFlag'], bikeBuyer\_1['Age'], 1)) (np.unique(bikeBuyer\_1['HouseOwnerFlag'])),color='red')

**Actual Dataframe:**

Graphical user interface, application

Description automatically generated

**BikeBuyer0:**

Graphical user interface, application, Word

Description automatically generated

**BikeBuyer 1:**

Graphical user interface, application, Word

Description automatically generated