Predicting the chain restaurant landscape in the United States based on County Sociodemographical Data from US Census

Importing Libraries and initial data

In [1]:
```
# Instalation of Pandas Profiling only for the firs time
#pip install ydata-profiling
```

In [2]:
```
# Importing the libraries needed Pandas and Numpy
import pandas as pd
import numpy as np

#Visualization and Profiling Libraries
import matplotlib.pyplot as plt
import seaborn as sns
import os
from ydata_profiling import ProfileReport

#General Libraries for Model Performance and Split
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_classification
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay
from dmba import classificationSummary

#Logistic Regression Library
from sklearn.linear_model import LogisticRegression

#Libraries for NN
from sklearn.neural_network import MLPClassifier

#Libraries for Chi2
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_selection import chi2
```

In [3]:
```
# Importing origial datasets
df1 = pd.read_csv("chainness_point_2021_part1.csv")
df2 = pd.read_csv("chainness_point_2021_part2.csv")
df3 = pd.read_csv("chainness_point_2021_part3.csv")
```

In [4]:
```
# Concat the dataset into one table
df = pd.concat([df1,df2,df3])
```

Exploratory Data Analysis

In [5]:
```
#Printing dataframe shape
print('Number of instances: ', df.shape[0])
print('Number of variables: ', df.shape[1])
```

```
Number of instances:  705621
Number of variables:  14
```

In [6]:
```
#Sampling dataframe
df.sample(5)
```

Out[6]:

	RestaurantName	Cuisine	OpenHours	State	CNTY_GEOID	CNTY_NAME	UA_GEOID	UA_NAME
94274	Wendy's	American	Sun Thu 630 AM 100 AM | Fri Sat 630 AM 200 AM	FL	12031	Duval	42346.0	Jacksonville, FL
167076	Arps	Restaurant	Tue Sun 1100 AM 200 PM	CO	8079	Mineral	NaN	NaN
214139	Cohens Bagel Company	American Cafe Deli Soups	Mon Fri 600 AM 300 PM | Sat Sun 700 AM 200 PM	CT	9009	New Haven	62407.0	New Haven, CT
235175	Ginos Pizza	Italian	Sun Sat 1100 AM 900 PM	NY	36061	New York	63217.0	New York--Newark, NY--NJ--CT
35593	McDonald's	Fast Food	Sun 600 AM 1100 PM | Mon 700 AM 1200 AM | Tu...	MT	30047	Lake	70588.0	Polson, MT

Understanding the variables

- RestaurantName: Restaurant name (processed)
- Cusine: Restaurant cuisine (raw)
- OpenHours: Restaurant's open hours (raw)
- State: State (raw)
- CNTY_GEOID: County GEOID, which can be joined with other datasets (processed)
- CNTY_NAME: The name of the county where the restaurant is located (processed)
- UA_GEOID: Urban area GEOID, which can be joined with other datasets (processed)
- UA_NAME: The name of the urban area where the restaurant is located (processed)
- MSA_GEOID: Metropolitan statistical area GEOID, which can be joined with other datasets (processed)
- MSA_NAME: The name of the metropolitan statistical area where the restaurant is located (processed)
- Lon: The longitude of the restaurant, projected to WGS84, crs=4326 (processed)
- Lat: The latitude of the restaurant, projected to WGS84, crs=4326 (processed)
- Frequency: The frequency of the restaurant (processed)
- isChain: A binary indicator that is 1 if the restaurant frequency > 5 else 0 (processed)

Data Cleaning and Formatting

Treating Null Values

In [7]:

```
# Verifying data type and variables with null values

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 705621 entries, 0 to 225620
Data columns (total 14 columns):
 #   Column          Non-Null Count   Dtype  
---  ------          --------------   -----  
 0   RestaurantName  705621 non-null  object 
 1   Cuisine         705607 non-null  object 
 2   OpenHours       463511 non-null  object 
 3   State           705621 non-null  object 
 4   CNTY_GEOID      705621 non-null  int64  
 5   CNTY_NAME       705621 non-null  object 
 6   UA_GEOID        631864 non-null  float64
 7   UA_NAME         631864 non-null  object 
 8   MSA_GEOID       613885 non-null  float64
 9   MSA_NAME        613885 non-null  object 
 10  Lon             705621 non-null  float64
 11  Lat             705621 non-null  float64
 12  Frequency       705621 non-null  int64  
 13  isChain         705621 non-null  int64  
dtypes: float64(4), int64(3), object(7)
memory usage: 80.8+ MB
```

The variables with null values are: 'OpenHours', 'UA_GEOID', 'UA_NAME', 'MSA_GEOID', 'MSA_NAME'

In [8]:
```
#Finding amount of null values per column
df_na = pd.DataFrame(df.isnull().sum())
df_na['Qty_NullValues']= df.isnull().sum()
```

In [9]:
```
#Finfing percentage of null values per column
df_na['Pct_NullValues'] = df_na['Qty_NullValues']/df.shape[0]
df_na = df_na.iloc[:,1:3]
df_na
```

Out[9]:

	Qty_NullValues	Pct_NullValues
RestaurantName	0	0.000000
Cuisine	14	0.000020
OpenHours	242110	0.343116
State	0	0.000000
CNTY_GEOID	0	0.000000
CNTY_NAME	0	0.000000
UA_GEOID	73757	0.104528
UA_NAME	73757	0.104528
MSA_GEOID	91736	0.130007
MSA_NAME	91736	0.130007
Lon	0	0.000000
Lat	0	0.000000
Frequency	0	0.000000
isChain	0	0.000000

The table above shows that over 34% of variable 'OpenHours' is null value. Other variables showed a maximum of 13%.

- The cleaning will be to drop the columns 'OpenHours' and erase the rows with null values

In [10]:
```
# Drop the columns 'OpenHours' since it contans null values and it is not relevant to the ana

df_cleaned = df.drop('OpenHours', axis=1)
```

In [11]:
```
# Drop rows with at least one null value.

df_cleaned = df_cleaned.dropna()
df_cleaned.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 575822 entries, 0 to 225619
Data columns (total 13 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   RestaurantName  575822 non-null  object
 1   Cuisine         575822 non-null  object
 2   State           575822 non-null  object
 3   CNTY_GEOID      575822 non-null  int64
 4   CNTY_NAME       575822 non-null  object
 5   UA_GEOID        575822 non-null  float64
 6   UA_NAME         575822 non-null  object
 7   MSA_GEOID       575822 non-null  float64
 8   MSA_NAME        575822 non-null  object
 9   Lon             575822 non-null  float64
 10  Lat             575822 non-null  float64
 11  Frequency       575822 non-null  int64
 12  isChain         575822 non-null  int64
dtypes: float64(4), int64(3), object(6)
memory usage: 61.5+ MB
```

In [12]:
```
print('Number of rows after cleaning null values: ', df_cleaned.shape[0])
```

```
Number of rows after cleaning null values:  575822
```

Treating Duplicated Rows

In [13]:
```
# Checking for duplicated rows

df_cleaned.duplicated().sum()
```

Out[13]:
```
37
```

In [14]:
```
# Droping duplicated rows

df_cleaned = df_cleaned.drop_duplicates()
```

In [15]:
```
print('Number of rows after cleaning duplicates: ', df_cleaned.shape[0])
```

```
Number of rows after cleaning duplicates:  575785
```

Pandas Profiling for the EDA

In [16]:
```
profile = ProfileReport(df_cleaned, title="EDA on U.S. Restaurants", explorative = True)
profile
```

```
Summarize dataset:   0%|          | 0/5 [00:00<?, ?it/s]
Generate report structure:   0%|          | 0/1 [00:00<?, ?it/s]
Render HTML:   0%|          | 0/1 [00:00<?, ?it/s]
```

Overview

Dataset statistics

Number of variables	13
Number of observations	575785
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	269.1 MiB
Average record size in memory	490.1 B

Variable types

Categorical	7
Numeric	6

Alerts

`RestaurantName` has a high cardinality: 291072 distinct values	**High cardinality**
`Cuisine` has a high cardinality: 18323 distinct values	**High cardinality**
`State` has a high cardinality: 51 distinct values	**High cardinality**

Out[16]:

In [17]:
```
profile.to_file("RestaurantsEDA.html")
```

```
Export report to file:   0%|          | 0/1 [00:00<?, ?it/s]
```

Formating Clean DataFrame

In [18]:
```
#Loding file with state name and achronym
states = pd.read_csv("states.csv")
```

In [19]:
```
#Left join of df_clean with states
df_cleaned=df_cleaned.merge(states, left_on='State', right_on='Postal')
```

In [20]:
```
#Renaming Columns
df_cleaned.rename(
```

```
    columns={"State_x": "Postal","State_y": "State"},
    inplace=True)
```

In [21]:
```
#Drop non-useful columns
df_cleaned= df_cleaned.drop(columns=['Postal'])
```

In [22]:
```
# Using + operator to combine two columns to create key to join tables later
df_cleaned['CountyKey'] = df_cleaned['CNTY_NAME'] + df_cleaned['State']
```

Unsupervised Segmentation to Define "Chainess" Buckets

Data Formatting

In [23]:
```
#Renaing df_cleaned for next section
df=df_cleaned
```

In [24]:
```
#Assing id by conting rows
df= df.assign(id=range(len(df)))

# Shift column id to first position
first_column = df.pop('id')
df.insert(0, 'id', first_column)
```

In [25]:
```
#Examining variable types for each columns
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 575785 entries, 0 to 575784
Data columns (total 15 columns):
 #   Column          Non-Null Count   Dtype  
---  ------          --------------   -----  
 0   id              575785 non-null  int64  
 1   RestaurantName  575785 non-null  object 
 2   Cuisine         575785 non-null  object 
 3   CNTY_GEOID      575785 non-null  int64  
 4   CNTY_NAME       575785 non-null  object 
 5   UA_GEOID        575785 non-null  float64
 6   UA_NAME         575785 non-null  object 
 7   MSA_GEOID       575785 non-null  float64
 8   MSA_NAME        575785 non-null  object 
 9   Lon             575785 non-null  float64
 10  Lat             575785 non-null  float64
 11  Frequency       575785 non-null  int64  
 12  isChain         575785 non-null  int64  
 13  State           575785 non-null  object 
 14  CountyKey       575785 non-null  object 
dtypes: float64(4), int64(4), object(7)
memory usage: 70.3+ MB
```

In [26]:
```
#Removing sub catefories from Cuisine Column
df['CuisineShort'] = df['Cuisine'].str.split(' ').str[0]
```

In [27]:
```
#Using drop() to delete rows based on column value
indexCuis = df[(df['CuisineShort']== 'Restaurant')].index
df.drop(indexCuis , inplace=True)
```

In [28]:
```
# creating list of restaurant types to keep as it is.
allowed_cuisines = ['American','Mexican','Pizza','Italian','Chinese','Cafe','Fast','Japanese'
# converting the rest to 'Other'
df.loc[~df['CuisineShort'].isin(allowed_cuisines), 'CuisineShort'] = 'Other'
```

In [29]:
```
#Renaming Columns
df.rename(
```

```
    columns={"Fast": "Fast Food", 'Middle':'Middle Eastern'},
    inplace=True)
```

In [30]:
```
#Loading Census Data and Walkability Index
df_census = pd.read_excel('SocioDemo.xlsx')
df_walk = pd.read_excel('Walkability Index.xlsx')
```

Chainess Variables Computation

The 'Chainess" of a County is derived from:

1. Avergae Replicability = Average of the 'Frequency' column. Indicates on average how many times one same restaurant has been replicated in a given county.
2. isChain Proportion = Number of Chains (where isChain = 1) / Total Restaurants (where isChain = 1 OR 0) by County. Indicates the percentage of restaurants that are chains in a given county.
3. Cuisine Proportion = Number of Restaurants Per Cuisine / Total Restaurants. Indicates the percentage of restaurants that belong to a cuisine in a given county.

Average Replicability

In [31]:
```
#Calculate avergae frequency of a restaurant by County
df_freq=df.groupby(['CNTY_GEOID'])['Frequency'].agg(['mean'])
```

In [32]:
```
#Turn multi index into columns
df_freq = df_freq.reset_index()
```

In [33]:
```
#Left merge of df and df_freq on CNTY_GEOID
df=df.merge(df_freq, left_on='CNTY_GEOID', right_on='CNTY_GEOID')
```

In [34]:
```
#Renaming Columns
df.rename(
    columns={"mean": "AvgRepl"},
    inplace=True)
```

In [35]:
```
# Checking for Outliers

# Count the number of outliers using IQR method
Q1 = df['AvgRepl'].quantile(0.25)
Q3 = df['AvgRepl'].quantile(0.75)
IQR = Q3 - Q1
outliers = df[(df['AvgRepl'] < (Q1 - 1.5 * IQR)) | (df['AvgRepl'] > (Q3 + 1.5 * IQR))]
num_outliers = len(outliers)
num_outliers
```

Out[35]:
```
10113
```

In [36]:
```
# Remove outliers using IQR method
df = df[(df['AvgRepl'] >= (Q1 - 1.5 * IQR)) & (df['AvgRepl'] <= (Q3 + 1.5 * IQR))]
```

isChain Proportion

In [37]:
```
#Calculate the isChain proportion by County
df_isch=df.groupby(['CNTY_GEOID'])['isChain'].agg(['sum','count'])
```

In [38]:
```
#Turn multi index into columns
df_isch = df_isch.reset_index()
```

In [39]:
```
#Calculating the proportion of restaurants that are chains
df_isch['isChainProp'] = df_isch['sum']/df_isch['count']
```

In [40]:
```
df=df.merge(df_isch, left_on='CNTY_GEOID', right_on='CNTY_GEOID')
```

In [41]:
```
#Drop non-useful columns
df= df.drop(columns=['sum','count'])
```

In [42]:
```
# Checking for Outliers

# Count the number of outliers using IQR method
Q1 = df['isChainProp'].quantile(0.25)
Q3 = df['isChainProp'].quantile(0.75)
IQR = Q3 - Q1
outliers = df[(df['isChainProp'] < (Q1 - 1.5 * IQR)) | (df['isChainProp'] > (Q3 + 1.5 * IQR))
num_outliers = len(outliers)
num_outliers
```

Out[42]:
```
447
```

In [43]:
```
# Remove outliers using IQR method
df = df[(df['isChainProp'] >= (Q1 - 1.5 * IQR)) & (df['isChainProp'] <= (Q3 + 1.5 * IQR))]
```

Cusine Replicability

In [44]:
```
#Calculate avergae frequency of a restaurant by County
df_freq=df.groupby(['CNTY_GEOID', 'CuisineShort'])['Frequency'].agg(['mean'])
```

In [45]:
```
#Turn multi index into columns
df_freq = df_freq.reset_index()
```

In [46]:
```
# Using + operator to combine two columns to create key to join tables later
df_freq['ConcatKey'] = df_freq['CNTY_GEOID'].astype(str) + df_freq['CuisineShort']
df['ConcatKey'] = df['CNTY_GEOID'].astype(str) + df['CuisineShort']
```

In [47]:
```
#Left join of df with df_cuisine using the ConcatKey above
df=df.merge(df_freq, left_on='ConcatKey', right_on='ConcatKey')
```

In [48]:
```
#Drop non-useful columns
df= df.drop(columns=['CNTY_GEOID_y', 'CuisineShort_y', 'ConcatKey'])
```

In [49]:
```
#Renaming Columns
df.rename(
    columns={"CuisineShort_x": "CuisineShort", "CNTY_GEOID_x": "CNTY_GEOID", "mean":"CuisRepl
    inplace=True)
```

In [50]:
```
# Checking for Outliers

# Count the number of outliers using IQR method
Q1 = df['CuisRepl'].quantile(0.25)
Q3 = df['CuisRepl'].quantile(0.75)
IQR = Q3 - Q1
outliers = df[(df['CuisRepl'] < (Q1 - 1.5 * IQR)) | (df['CuisRepl'] > (Q3 + 1.5 * IQR))]
num_outliers = len(outliers)
num_outliers
```

Out[50]:
```
20885
```

In [51]:
```
# Remove outliers using IQR method
df = df[(df['CuisRepl'] >= (Q1 - 1.5 * IQR)) & (df['CuisRepl'] <= (Q3 + 1.5 * IQR))]
```

Grouping Data By Cuisines

In [52]:
```
# Using pandas qcut() to assign each row to a latitude and longitude area
df['LatBin'] = pd.qcut(df['Lat'], q=25, labels=False)
df['LonBin'] = pd.qcut(df['Lon'], q=25, labels=False)
```

In [53]:
```
#Drop non-useful columns
df= df.drop(columns=['id', 'RestaurantName', 'Cuisine', 'isChain', 'Frequency','Lat', 'Lon'])
```

In [54]:
```
# Checking for duplicated rows

df.duplicated().sum()
```

Out[54]:
```
326436
```

In [55]:
```
# Droping duplicated rows
df = df.drop_duplicates()
```

In [56]:
```
df = df.assign(row_number=range(len(df)))
```

In [57]:
```
#Renaming Columns
df.rename(
    columns={"row_number": "id"},inplace=True)
```

Assigning Chainability

In [58]:
```
#Subsetting Data to include only variables defining Chainess
df_clus = df[['isChainProp', 'AvgRepl', "CuisRepl"]]
```

In [59]:
```
# Computing the ideal value of k.
#The ideal value of k is where the curve "bends" or the inflection point of the curve.
#In this case the ideal value of k is 3.
from sklearn.cluster import KMeans
wcss = []
for k in range(1,11):
    kmeans = KMeans(n_clusters=k, init="k-means++")
    kmeans.fit(df_clus.iloc[:,1:])
    wcss.append(kmeans.inertia_)
plt.figure(figsize=(12,6))
plt.grid()
plt.plot(range(1,11),wcss, linewidth=2, color="red", marker ="8")
plt.xlabel("K Value")
plt.xticks(np.arange(1,11,1))
plt.ylabel("WCSS")
plt.show()
```

```
C:\Users\Megan\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The
default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` exp
licitly to suppress the warning
  warnings.warn(
C:\Users\Megan\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The
default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` exp
licitly to suppress the warning
  warnings.warn(
C:\Users\Megan\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The
default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` exp
licitly to suppress the warning
  warnings.warn(
C:\Users\Megan\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The
default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` exp
licitly to suppress the warning
  warnings.warn(
C:\Users\Megan\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The
default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` exp
licitly to suppress the warning
  warnings.warn(
C:\Users\Megan\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The
default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` exp
licitly to suppress the warning
  warnings.warn(
C:\Users\Megan\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The
default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` exp
licitly to suppress the warning
  warnings.warn(
C:\Users\Megan\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The
default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` exp
licitly to suppress the warning
  warnings.warn(
C:\Users\Megan\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The
default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` exp
licitly to suppress the warning
  warnings.warn(
C:\Users\Megan\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The
default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` exp
licitly to suppress the warning
  warnings.warn(
C:\Users\Megan\AppData\Local\Temp\ipykernel_21972\4129201520.py:16: UserWarning: Matplotlib i
s currently using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so c
annot show the figure.
  plt.show()
```

In [60]:

```
# Applying K Means Clustering with k = 3
km = KMeans(n_clusters=3)
clusters = km.fit_predict(df_clus.iloc[:,1:])
df_clus["Clusters"] = clusters
df_clus
```

```
C:\Users\Megan\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The
default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` exp
licitly to suppress the warning
  warnings.warn(
C:\Users\Megan\AppData\Local\Temp\ipykernel_21972\620799678.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guid
e/indexing.html#returning-a-view-versus-a-copy
  df_clus["Clusters"] = clusters
```

Out[60]:

	isChainProp	AvgRepl	CuisRepl	Clusters
19	0.465990	1204.107614	1245.586777	2
24	0.465990	1204.107614	1245.586777	2
29	0.465990	1204.107614	1245.586777	2
362	0.465990	1204.107614	1245.586777	2
431	0.465990	1204.107614	582.325581	0
...	...	...	...	...
373320	0.190476	1410.714286	1.125000	0
373321	0.190476	1410.714286	1.125000	0
373328	0.190476	1410.714286	2.000000	0
373329	0.190476	1410.714286	1.000000	0
373330	0.190476	1410.714286	1.000000	0

26010 rows × 4 columns

In [61]:

```
#Plotting clusters against the three defining variables
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

fig = plt.figure(figsize=(20,10))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(df_clus.CuisRepl[df_clus.Clusters == 0], df_clus["AvgRepl"][df_clus.Clusters == 0]
ax.scatter(df_clus.CuisRepl[df_clus.Clusters == 1], df_clus["AvgRepl"][df_clus.Clusters == 1]
ax.scatter(df_clus.CuisRepl[df_clus.Clusters == 2], df_clus["AvgRepl"][df_clus.Clusters == 2]
ax.scatter(df_clus.CuisRepl[df_clus.Clusters == 3], df_clus["AvgRepl"][df_clus.Clusters == 3]

ax.view_init(30, 185)
plt.xlabel("Cuisine Proportion")
plt.ylabel("Average Replicability")
ax.set_zlabel('Chain Proportion')
plt.show()
```

```
C:\Users\Megan\AppData\Local\Temp\ipykernel_21972\1159020628.py:18: UserWarning: Matplotlib i
s currently using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so c
annot show the figure.
  plt.show()
```

In [62]:

```
#Box Plot of Cuisine Proportion by Cluster
#It does not seem to be a remarkable difference between clusters.
CuisPlot = sns.boxplot(x='Clusters', y='CuisRepl', data=df_clus)
CuisPlot
```

Out[62]:

```
<Axes3DSubplot: xlabel='Clusters', ylabel='CuisRepl', zlabel='Chain Proportion'>
```

In [63]:

```
#Box Plot of Average Replicability by Cluster
#There seems to be a difference between clusters.
AvgRepl = sns.boxplot(x='Clusters', y='AvgRepl', data=df_clus)
AvgRepl
```

Out[63]:

```
<Axes3DSubplot: xlabel='Clusters', ylabel='AvgRepl', zlabel='Chain Proportion'>
```

In [64]:

```
#Box Plot of isChain Proportion by Cluster
#There seems to be a difference between clusters.
isChainPlot = sns.boxplot(x='Clusters', y='isChainProp', data=df_clus)
isChainPlot
```

Out[64]: <Axes3DSubplot: xlabel='Clusters', ylabel='isChainProp', zlabel='Chain Proportion'>

In [65]:
```
#Deriving id column
df_clus = df_clus.assign(row_number=range(len(df)))
```

In [66]:
```
#Renaming Columns
df_clus.rename(
    columns={"row_number": "id"},inplace=True)
```

In [67]:
```
#Drop non-useful columns
df_clus= df_clus.drop(columns=['CuisRepl','AvgRepl', 'isChainProp'])
```

In [68]:
```
#Join cluster data with original data set
df=df.merge(df_clus, left_on='id', right_on='id')
```

In [69]:
```
#Name cluster with low, medium and high
df['Chainess'] = np.where(df['Clusters']==1, 'High',
                          np.where(df['Clusters']==0, 'Medium','Low'))
```

In [70]:
```
#Calculate avergae frequency of a restaurant by County
df_balance=df.groupby(['Chainess'])['id'].agg(['count'])
df_balance
```

Out[70]:

Chainess	count
High	2224
Low	4591
Medium	19195

US Census Data, Formatting and Variable Selection

In [71]:
```
#Left join of with Census Data
df=df.merge(df_census, left_on='CountyKey', right_on='CountyKey')
```

In [72]:
```
#Left join of with Walkability Index
df=df.merge(df_walk, left_on='CNTY_GEOID', right_on='GEOID')
```

In [73]:
```
#Finding amount of null values per column
df_na = pd.DataFrame(df.isnull().sum())
df_na['Qty_NullValues']= df.isnull().sum()
df_na
```

Out[73]:

	0	Qty_NullValues
CNTY_GEOID	0	0
CNTY_NAME	0	0
UA_GEOID	0	0
UA_NAME	0	0
MSA_GEOID	0	0
MSA_NAME	0	0
State	0	0
CountyKey	0	0
CuisineShort	0	0
AvgRepl	0	0
isChainProp	0	0
CuisRepl	0	0
LatBin	0	0
LonBin	0	0
id	0	0
Clusters	0	0
Chainess	0	0
Median age (years)	0	0
Sex ratio (males per 100 females)	0	0
Age dependency ratio	0	0
Old-age dependency ratio	0	0
Child dependency ratio	0	0
Population	22	22
Gini Index	0	0
Income Per Capita	252	252
Density	1049	1049
GEOID	0	0
NatWalkInd	0	0

In [74]:

```
# Drop rows with at least one null value.
df = df.dropna()
```

In [75]:

```
#Change NatWalkInd type to numerical
df = df.astype({'NatWalkInd':'float'})
```

In [76]:

```
#Creating Bins for all numerical Predictors

# Convert MedianAge to categorical
df['AgeRangeBin'] = pd.cut(df['Median age (years)'], [0, 39, 59, float('inf')], labels=['Youn
#Source: https://www.researchgate.net/figure/Age-intervals-and-age-groups_tbl1_228404297

# Bin Walkability
df['NatWalkIndBin'] = pd.cut(df['NatWalkInd'], [0,5.75, 10.5, 15.25, float('inf')], right=Fal
#Source: https://www.epa.gov/sites/default/files/2021-06/documents/national_walkability_index
```

```
# Bin Income
df['IncomeBin'] = pd.cut(df['Income Per Capita'], [0,26100, 78300, float('inf')], right=False
#Source: https://money.usnews.com/money/personal-finance/family-finance/articles/where-do-i-f
#Divided by 2 cause it is just 1 person

# Bin Gini
df['GiniBin'] = pd.cut(df['Gini Index'], [0,.42,.54, float('inf')], right=False, labels=['Low
#Source: https://www.researchgate.net/figure/Map-of-national-income-Gini-coefficients-reporte
```

In [77]:
```
#Checking for new columns and null values
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 24834 entries, 0 to 26020
Data columns (total 32 columns):
 #   Column                           Non-Null Count  Dtype
---  ------                           --------------  -----
 0   CNTY_GEOID                       24834 non-null  object
 1   CNTY_NAME                        24834 non-null  object
 2   UA_GEOID                         24834 non-null  float64
 3   UA_NAME                          24834 non-null  object
 4   MSA_GEOID                        24834 non-null  float64
 5   MSA_NAME                         24834 non-null  object
 6   State                            24834 non-null  object
 7   CountyKey                        24834 non-null  object
 8   CuisineShort                     24834 non-null  object
 9   AvgRepl                          24834 non-null  float64
 10  isChainProp                      24834 non-null  float64
 11  CuisRepl                         24834 non-null  float64
 12  LatBin                           24834 non-null  int64
 13  LonBin                           24834 non-null  int64
 14  id                               24834 non-null  int64
 15  Clusters                         24834 non-null  int32
 16  Chainess                         24834 non-null  object
 17  Median age (years)               24834 non-null  float64
 18  Sex ratio (males per 100 females) 24834 non-null  float64
 19  Age dependency ratio             24834 non-null  float64
 20  Old-age dependency ratio         24834 non-null  float64
 21  Child dependency ratio           24834 non-null  float64
 22  Population                       24834 non-null  float64
 23  Gini Index                       24834 non-null  float64
 24  Income Per Capita                24834 non-null  float64
 25  Density                          24834 non-null  float64
 26  GEOID                            24834 non-null  object
 27  NatWalkInd                       24834 non-null  float64
 28  AgeRangeBin                      24834 non-null  category
 29  NatWalkIndBin                    24834 non-null  category
 30  IncomeBin                        24834 non-null  category
 31  GiniBin                          24834 non-null  category
dtypes: category(4), float64(15), int32(1), int64(3), object(9)
memory usage: 5.5+ MB
```

In [78]:
```
#Saving Data set with Predictos and Bins to csv
df.to_csv('df_clusters.csv')
```

Variable Selection

Numerical Variables

In [79]:
```
#Subseting all possible numerical predictors
df_corr = df[['Median age (years)', 'Age dependency ratio','Old-age dependency ratio','Child
```

In [80]:
```
#Building Correlation Matrix
corr = df_corr.corr()
plt.rcParams['xtick.top'] = plt.rcParams['xtick.labeltop'] = True
```

```
plt.figure(figsize=(10, 9))
sns.heatmap(corr, xticklabels=corr.columns, yticklabels=corr.columns,annot=True,cmap="RdBu")
```

Out[80]: <AxesSubplot: >

Age Dependency Ratio and Old Dependency Ration have a high correlation among them and with median age. Therefore, they were removed from the predictors. Child Dependency Ratio was also removed beacuse it was deemed unrelevant to the output.

In [81]:
```
#Subseting all possible numerical predictors after removing highly correlated predictors
df_corr = df[['Median age (years)','Population','Income Per Capita', 'Gini Index','NatWalkInd
```

In [82]:
```
#Building Correlation Matrix after removing highly correlated predictors
corr = df_corr.corr()
plt.rcParams['xtick.top'] = plt.rcParams['xtick.labeltop'] = True
plt.figure(figsize=(9, 9))
sns.heatmap(corr, xticklabels=corr.columns, yticklabels=corr.columns,annot=True,cmap="RdBu")
```

Out[82]: <AxesSubplot: >

Categorical Variables

In [83]:
```
#Enconding Categorical Varibles
label_encoder = LabelEncoder()
df['ChainessChi'] = label_encoder.fit_transform(df['Chainess'])
df['CuisineChi'] = label_encoder.fit_transform(df['CuisineShort'])
df['AgeChi'] = label_encoder.fit_transform(df['AgeRangeBin'])
df['IncomeChi'] = label_encoder.fit_transform(df['IncomeBin'])
df['WalkChi'] = label_encoder.fit_transform(df['NatWalkIndBin'])
df['GiniChi'] = label_encoder.fit_transform(df['GiniBin'])
```

In [84]:
```
#Declaring X and y to apply the method
X = df[['CuisineChi','LatBin', 'LonBin','AgeChi','IncomeChi','WalkChi','GiniChi']]
y = df['ChainessChi']
```

In [85]:
```
# Running chi2 test
chi_scores = chi2(X,y)
chi_scores
```

Out[85]:
```
(array([3.27229658e+03, 9.79109711e+01, 4.10066076e+01, 8.68039078e+00,
        8.15128751e-01, 5.40064987e+01, 5.76813057e+00]),
 array([0.00000000e+00, 5.48154218e-22, 1.24602946e-09, 1.30339812e-02,
        6.65268623e-01, 1.87343148e-12, 5.59070223e-02]))
```

In [86]:
```
#Printing p-values
p_values = pd.Series(chi_scores[1],index = X.columns)
p_values.sort_values(ascending = False , inplace = True)
p_values
```

Out[86]:
```
IncomeChi      6.652686e-01
GiniChi        5.590702e-02
AgeChi         1.303398e-02
LonBin         1.246029e-09
WalkChi        1.873431e-12
LatBin         5.481542e-22
CuisineChi     0.000000e+00
dtype: float64
```

Income Per Capita has a p-Vaulue greater than 0.5 therefore it was removed from the predictors.

One Hot Econding (Dummies)

In [87]:
```
#Change variable type to categoy for numerical categories
df['LatBin'] = df['LatBin'].astype('category')
```

```
df['LonBin'] = df['LonBin'].astype('category')
```

In [88]:
```
#Assign predictors to X and outcome to Y
X = pd.get_dummies(df[['CuisineShort','LatBin', 'LonBin','AgeRangeBin','NatWalkIndBin','GiniB
y = df['Chainess'].astype('category')
classes = list(y.cat.categories)
```

In [89]:
```
#Create dataframe for Logistic Regression and NN where predictors are dummies bu outcome is n
df_ml = X.assign(Chainess = y)
```

In [90]:
```
#Save dataframe to csv for Logistic Regression and NN
df_ml.to_csv('df_dummies.csv')
```

In [91]:
```
#Change all variables to dummies (even outcome) for apriori
df_priori = pd.get_dummies(df[['CuisineShort','LatBin', 'LonBin','AgeRangeBin','NatWalkIndBin
```

In [92]:
```
#Save dataframe to csv for Apriori
df_priori.to_csv('df_apriori.csv')
```

Logistic Regression Algorithm

In [93]:
```
#Asign new name to dataframe
df_logistic = df_ml
```

In [94]:
```
#Replacing the space between the words with underscores
df_logistic.columns = [s.strip().replace(' ','_') for s in df_logistic.columns]
```

In [95]:
```
#Selecting Predictors and Outcome
predictors = df_logistic.columns.drop('Chainess')
outcome = ['Chainess']

X = df_logistic[predictors]
y = df_logistic[outcome]
```

In [96]:
```
#Data Partition with test size = 40%
train_X, valid_X, train_y, valid_y = train_test_split(X, y, test_size=0.6, random_state=42)
```

In [97]:
```
# Perform Logistic regression balancing classes
log_reg = LogisticRegression(solver='newton-cg', class_weight='balanced')
log_reg.fit(train_X,train_y)
```

```
C:\Users\Megan\anaconda3\lib\site-packages\sklearn\utils\validation.py:1143: DataConversionWa
rning: A column-vector y was passed when a 1d array was expected. Please change the shape of
y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

Out[97]:
```
▾ LogisticRegression
LogisticRegression(class_weight='balanced', solver='newton-cg')
```

In [98]:
```
# Make prediction using the model
# perform prediction using the test dataset

y_train_pred = log_reg.predict(train_X)
y_valid_pred = log_reg.predict(valid_X)
```

In [99]:
```
# Print model performnance
print('Training')
classificationSummary(train_y, y_train_pred)
ConfusionMatrixDisplay.from_predictions(train_y,y_train_pred)

print('Validation')
```

```
classificationSummary(valid_y, y_valid_pred)
ConfusionMatrixDisplay.from_predictions(valid_y,y_valid_pred)
```

```
Training
Confusion Matrix (Accuracy 0.8429)

       Prediction
Actual    0    1    2
     0  627  193   11
     1  501 1154   73
     2  276  506 6592
Validation
Confusion Matrix (Accuracy 0.8349)

       Prediction
Actual    0    1    2
     0  913  343   18
     1  849 1649  123
     2  391  736 9879
```

Out[99]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x17f0d7cc910>
```

Neural Network Algorithm

In [100…

```
# train neural network with 1 hidden layer and 2 nodes
clf = MLPClassifier(hidden_layer_sizes=(40), activation='logistic', solver='lbfgs',random_sta
clf.fit(train_X, train_y.values)
```

```
C:\Users\Megan\anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:1
096: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please
change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

Out[100]:

```
▾ MLPClassifier
MLPClassifier(activation='logistic', hidden_layer_sizes=40, max_iter=500,
              random_state=1, solver='lbfgs')
```

In [101…

```
clf.predict(X)
```

Out[101]:

```
array(['Low', 'Low', 'Low', ..., 'Medium', 'Medium', 'Medium'],
      dtype='<U6')
```

In [102…

```
# Print model perfomnance
#NN Model Evaluation
# training performance
print('Trainning Performance')
classificationSummary(train_y, clf.predict(train_X))

print('Validation Performance')
# validation performance
classificationSummary(valid_y, clf.predict(valid_X))
```

```
Trainning Performance
Confusion Matrix (Accuracy 0.9761)

       Prediction
Actual    0    1    2
     0  774   47   10
     1   36 1627   65
     2   12   67 7295
Validation Performance
Confusion Matrix (Accuracy 0.8815)

       Prediction
Actual     0     1     2
     0   792   345   137
     1   336  1858   427
     2   107   414 10485
```

In [103…

```python
# Network structure: Intercepts
print('Intercepts')
clf.intercepts_
```

```
Intercepts
```

Out[103]:

```
[array([-0.96678834,  1.62926058,  0.6739824 , -0.96242113,  1.84580545,
        -1.18395929,  3.15961567, -1.56521648, -0.95542004, -0.51106833,
        -4.8377949 ,  0.68768808, -1.33163431, -0.44774931,  2.52917504,
        -4.57446814, -2.9673629 , -2.37829618, -0.7401056 ,  1.81455465,
        -3.06373802, -0.89756441,  0.89820223, -1.51147016, -0.49069892,
        -0.22102081, -4.06644789,  1.59746883, -2.98520853, -0.83581247,
         0.07555291, -1.51373994,  2.82407911, -3.29736118, -1.69388914,
        -3.33679363,  3.99698921, -0.87010794, -0.25298331,  1.64008776]),
 array([-3.97043665,  4.54482594, -0.50623587])]
```

In [104…

```python
# Network structure: Coeficients
print('Weights')
clf.coefs_
```

```
Weights
```

```
Out[104]: [array([[-4.45998002,  4.29575099, -7.84300864, ..., -3.98691596,
          -7.06505631,  1.95319946],
         [ 2.6090922 ,  0.75235027,  0.69817825, ..., -2.40998985,
           1.95938617, -2.77420758],
         [-1.15958261,  0.2040192 ,  1.30555599, ...,  0.36491947,
           0.98467529, -0.67903691],
         ...,
         [ 5.60566671,  1.33197113,  4.37888581, ...,  3.96380909,
           0.53103083, -0.25499957],
         [-2.41389825, -0.7123228 , -3.31366462, ..., -2.9208934 ,
           1.05113024,  2.5515475 ],
         [-4.18496598,  0.9144798 , -0.30878971, ..., -1.84576345,
          -1.76815378, -0.4032556 ]]),
 array([[  3.84841714,  -8.56271384,   4.54903462],
         [-11.73368523,   6.2616555 ,   5.67957324],
         [ -1.23296394,  -5.84942949,   6.87109298],
         [  7.7321755 ,  -2.00771403,  -5.71101715],
         [  3.49768244,  -7.53128082,   3.58574588],
         [ -7.00950729,  -8.4985319 ,  15.51190717],
         [  7.03082107,  -9.07237988,   1.91006174],
         [ -4.66718178, -13.79892395,  18.41897952],
         [  8.21699852,  -6.11931692,  -2.10018185],
         [-10.0287156 ,   9.82047359,   0.53388128],
         [ 14.70789449,  -5.53890619,  -9.11900877],
         [ -5.46805631,  -5.26856467,  10.80771553],
         [  2.2925455 ,  -8.54833679,   6.46597218],
         [ -4.31335361,  -5.34757977,   9.12806295],
         [ -2.57874586,  -7.69486037,  10.23805766],
         [ 12.74425369,  -4.4292938 ,  -8.40301514],
         [  9.24515424,  -2.99983494,  -6.19382086],
         [ -2.28244449,   7.40311202,  -5.25067177],
         [-12.1471138 ,   2.56856211,   9.73924497],
         [ -6.39302083,  -4.64521151,  10.68126424],
         [ -0.4746253 ,  10.13081037, -10.02089311],
         [ -6.60942485,   8.84414305,  -2.03997436],
         [ -0.04579192,   9.36874762,  -9.50834084],
         [  0.70410472,   8.3871453 ,  -9.14295405],
         [ -7.72090477,   4.27436909,   3.93007779],
         [  0.37760332,  10.23574396, -10.57477978],
         [ 10.74717003, -12.51860152,   1.65643281],
         [ 11.55320644,  -4.95874351,  -6.556725  ],
         [  2.17900602, -13.5356795 ,  11.16520839],
         [  6.56858307,   4.62113798, -11.02582607],
         [-10.27448903,  -1.78804727,  12.10287416],
         [ -8.59885486,   1.77630484,   6.587719  ],
         [  3.69326234,   5.82105525,  -9.39667759],
         [-12.56901523,   8.04804204,   4.60176415],
         [ -1.04078869,   0.07374467,   0.61594524],
         [  8.29199348,  -1.0832589 ,  -7.30289539],
         [-14.3948715 ,   3.1061974 ,  10.83481663],
         [ -4.44236721,   9.84627527,  -5.31093242],
         [  6.87990081,  -9.15397994,   2.46534275],
         [  7.39402275,   4.6900488 , -12.09401545]])]
```

```
In [105…
# Prediction Probabilities
pd.concat([df,pd.DataFrame(clf.predict_proba(X))], axis=1)
```

Out[105]:

	CNTY_GEOID	CNTY_NAME	UA_GEOID	UA_NAME	MSA_GEOID	MSA_NAME	State	CountyKey
0	40143	Tulsa	88948.0	Tulsa, OK	46140.0	Tulsa, OK	Oklahoma	TulsaOklahoma
1	40143	Tulsa	88948.0	Tulsa, OK	46140.0	Tulsa, OK	Oklahoma	TulsaOklahoma
2	40143	Tulsa	18760.0	Collinsville, OK	46140.0	Tulsa, OK	Oklahoma	TulsaOklahoma
3	40143	Tulsa	82360.0	Skiatook, OK	46140.0	Tulsa, OK	Oklahoma	TulsaOklahoma
4	40143	Tulsa	88948.0	Tulsa, OK	46140.0	Tulsa, OK	Oklahoma	TulsaOklahoma
...	...	...	...	...	...	...	...	...
24623	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
24624	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
24625	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
24626	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
24627	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

25987 rows × 41 columns

https://journals.sagepub.com/doi/full/10.1177/23998083211014896

https://www.sciencedirect.com/science/article/abs/pii/S0143622812000811

https://towardsdatascience.com/customer-segmentation-using-k-means-clustering-d33964f238c3#:~:text=The%20goal%20of%20K%20means,the%20revenue%20of%20the%20company.

In [ ]:

Apriori and Association Rules

In [106…

```
import pandas as pd
from mlxtend.frequent_patterns import apriori
from mlxtend.preprocessing import TransactionEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

In [107…

```
# Load your dataset with dummy variables
df = pd.read_csv('df_dummies.csv')
```

In [108…

```
# Convert dummy variables to binary transactional data
transactional_data = []

for index, row in df.iterrows():
    transaction = [col for col in df.columns if row[col] == 1]
    transactional_data.append(transaction)
```

In [109…

```
# Apriori for Frequent Itemset Mining
# Using TransactionEncoder to transform transactional data to binary format

te = TransactionEncoder()
te_ary = te.fit_transform(transactional_data)
binary_df = pd.DataFrame(te_ary, columns=te.columns_)
```

In [ ]:

In [110…
```
# finding frequent itemsetsv
frequent_itemsets = apriori(binary_df, min_support=0.1, use_colnames=True)
```

In [111…
```
# Extracting association rules from frequent itemsets
from mlxtend.frequent_patterns import association_rules

association_rules_df = association_rules(frequent_itemsets, metric="confidence", min_threshol
```

In [112…
```
# new features from association rules
df['association_rule'] = False

for index, row in association_rules_df.iterrows():
    antecedents = set(row['antecedents'])
    consequents = set(row['consequents'])
    for i, transaction in enumerate(transactional_data):
        if antecedents.issubset(transaction) and consequents.issubset(transaction):
            df.at[i, 'association_rule'] = True
```

In [113…
```
# CART Classification Model
# Prepare X and y for classification
X = df.drop(['association_rule', 'Chainess'], axis=1)  # Replace 'target_variable' with the n
y = df['Chainess']  # Replace 'target_variable' with the name of your target variable
```

In [114…
```
# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [115…
```
# Train a CART
clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)
```

Out[115]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

In [116…
```
# Running the model on test data
y_pred = clf.predict(X_test)
```

In [117…
```
# Evaluate the model
accuracy = accuracy_score(y_train, clf.predict(X_train))
print(f'Train Accuracy: {accuracy}')

# validation performance
accuracy = accuracy_score(y_test, y_pred)
print(f'Validation Accuracy: {accuracy}')
```

```
Train Accuracy: 1.0
Validation Accuracy: 0.9323535333199114
```