Part 1: Data Cleansing

Code for pre-processing steps performed

bazalewski_capstone_cleaning

April 28, 2022

[1]:
```
import pandas as pd
pd.set_option('display.max_rows', 1000)
```

1 Sales Data

[2]:
```
#import sales csv
sales = pd.read_csv('new_business.csv')

#make names uppercase
sales['cleaned_name']=sales['Client Name (first L)'].str.upper().str.strip().
↪str.replace(',','').str.replace(' ','').str.replace('.','')
```

```
C:\Users\julie\anaconda3\lib\site-packages\ipykernel_launcher.py:5:
FutureWarning: The default value of regex will change from True to False in a
future version. In addition, single character regular expressions will*not* be
treated as literal strings when regex=True.
  """
```

[3]:
```
#import contacts csv
contacts = pd.read_csv('contacts_confidential.csv')

#remove those without zip codes
contacts_w_zip = contacts[contacts['Merged Zip'] != ' '].reset_index()

contacts_w_zip['cleaned_name'] = contacts_w_zip['Titleized First L'].str.
↪upper().str.strip().str.replace(',','').str.replace(' ','').str.replace('.
↪','')
```

```
C:\Users\julie\anaconda3\lib\site-packages\ipykernel_launcher.py:7:
FutureWarning: The default value of regex will change from True to False in a
future version. In addition, single character regular expressions will*not* be
treated as literal strings when regex=True.
  import sys
```

[4]:
```
#need to add another column for the cases when there is a couple name in the␣
↪sales data ("AND"). Add both directions
```

```
contacts_w_zip['combined name'] = contacts_w_zip['Last Name'].str.strip().
 ↪str[0] + contacts_w_zip['First Name1'].str.strip() + 'AND' +␣
 ↪contacts_w_zip['First Name2'].str.strip()
contacts_w_zip['combined name 2'] = contacts_w_zip['Last Name'].str.strip().
 ↪str[0] + contacts_w_zip['First Name2'].str.strip() +  'AND' +␣
 ↪contacts_w_zip['First Name1'].str.strip()
```

[5]:
```
#determine if there are still any duplicate entries
df = contacts_w_zip.groupby(['Titleized First L','Last Name','Merged Zip']).
 ↪agg([('total', 'count')]).reset_index()
filter_df = df[['Titleized First L', 'Last Name', 'First Name1', 'Merged Zip']]
filter_df[filter_df[('First Name1','total')] > 1].reset_index()
```

[5]:
```
Empty DataFrame
Columns: [(index, ), (Titleized First L, ), (Last Name, ), (First Name1, total),
(Merged Zip, )]
Index: []
```

[6]:
```
#determine if there are still any duplicate entries
df = contacts_w_zip.groupby(['Titleized First L']).agg([('total', 'count')]).
 ↪reset_index()
filter_df = df[['Titleized First L', 'Last Name']]
#filter_df
filter_df[filter_df[('Last Name','total')]> 1].reset_index()
```

[6]:
```
Empty DataFrame
Columns: [(index, ), (Titleized First L, ), (Last Name, total)]
Index: []
```

1.1 Nicknames

[7]:
```
#import nicknames csv (https://github.com/carltonnorthern/
 ↪nickname-and-diminutive-names-lookup)
nicknames = pd.read_csv('names.csv')
nicknames = nicknames.apply(lambda x: x.astype(str).str.upper())
```

[8]:
```
nickname_df = contacts_w_zip.merge(nicknames, left_on='First Name1',␣
 ↪right_on='name').
 ↪drop(['name','nickname7','nickname8','nickname9','nickname10','nickname11','nickname13','ni
```

[9]:
```
nickname_df['nickname1'] = nickname_df['Last Name'].str[0] +␣
 ↪nickname_df['nickname1']
nickname_df['nickname2'] = nickname_df['Last Name'].str[0] +␣
 ↪nickname_df['nickname2']
nickname_df['nickname3'] = nickname_df['Last Name'].str[0] +␣
 ↪nickname_df['nickname3']
```

```
nickname_df['nickname4'] = nickname_df['Last Name'].str[0] +␣
↪nickname_df['nickname4']
```

1.2 Join Sales and Contact Data Frames

[10]:
```
df_join = sales.merge(contacts_w_zip, on='cleaned_name')
```

[11]:
```
df_combined_names1_join = sales.merge(contacts_w_zip[contacts_w_zip['combined␣
↪name'].str.contains('AND')], left_on='cleaned_name', right_on='combined␣
↪name')
df_combined_names1_join = df_combined_names1_join.
↪drop(['cleaned_name_y'],axis=1)
```

[12]:
```
df_combined_names2_join = sales.merge(contacts_w_zip[contacts_w_zip['combined␣
↪name'].str.contains('AND')], left_on='cleaned_name', right_on='combined name␣
↪2')
df_combined_names2_join = df_combined_names2_join.
↪drop(['cleaned_name_y'],axis=1)
```

[13]:
```
joined_df = pd.
↪concat([df_join,df_combined_names1_join,df_combined_names2_join]).
↪reset_index()
joined_df = joined_df.drop(['cleaned_name_x'],axis=1)
```

[14]:
```
#check for duplicates
joined_df = joined_df.drop_duplicates()
```

[15]:
```
df_nicknames1_joined = sales.merge(nickname_df, left_on='cleaned_name',␣
↪right_on='nickname1')
df_nicknames2_joined = sales.merge(nickname_df, left_on='cleaned_name',␣
↪right_on='nickname2')
df_nicknames3_joined = sales.merge(nickname_df, left_on='cleaned_name',␣
↪right_on='nickname3')
df_nicknames4_joined = sales.merge(nickname_df, left_on='cleaned_name',␣
↪right_on='nickname4')
```

[16]:
```
joined_df = joined_df.drop(['level_0'],axis=1)
joined_df = pd.concat([joined_df,df_nicknames1_joined]).reset_index()
joined_df = joined_df.drop(['level_0'],axis=1)
joined_df = pd.concat([joined_df,df_nicknames2_joined]).reset_index()
joined_df = joined_df.drop(['level_0'],axis=1)
joined_df = pd.concat([joined_df,df_nicknames3_joined]).reset_index()
joined_df = joined_df.drop(['level_0'],axis=1)
joined_df = pd.concat([joined_df,df_nicknames4_joined]).reset_index()
```

[17]:
```
joined_df_final = joined_df[['Initial Contact Month', 'Initial Contact Year',␣
↪'Referral Source', 'Practice Area', 'Fee', 'Cleaned Cities', 'State',␣
↪'Merged Zip']]
```

[18]:
```
joined_df_final['Fee'] = joined_df_final['Fee'].str.strip().str.replace(',','').
↪str.replace('$','').str.replace('-','').str.replace(' ','NaN')
joined_df_final['Fee'] = pd.to_numeric(joined_df_final['Fee'], errors='coerce')
```

```
C:\Users\julie\anaconda3\lib\site-packages\ipykernel_launcher.py:1:
FutureWarning: The default value of regex will change from True to False in a
future version. In addition, single character regular expressions will*not* be
treated as literal strings when regex=True.
  """Entry point for launching an IPython kernel.
C:\Users\julie\anaconda3\lib\site-packages\ipykernel_launcher.py:1:
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_guide/indexing.html#returning-a-view-versus-a-copy
  """Entry point for launching an IPython kernel.
C:\Users\julie\anaconda3\lib\site-packages\ipykernel_launcher.py:2:
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_guide/indexing.html#returning-a-view-versus-a-copy
```

[19]:
```
joined_df_final.columns = ['Contact Month', 'Contact Year', 'Referral',␣
↪'Practice Area', 'Fee', 'City', 'State', 'Zip']
```

[20]:
```
joined_df_final.to_csv('cleaned_sales.csv')
```

2 Calls Data

[21]:
```
#import calls csv
calls = pd.read_csv('call_list.csv')
```

[22]:
```
joined_calls_df = joined_df.merge(calls, left_on='Formatted Phone',␣
↪right_on='Phone Number', how="right")
```

[23]:
```
joined_calls_df_final = joined_calls_df[['Fee','Merged␣
↪Zip','City','State_y','Source','Duration (seconds)',
```

```
                                                'Start Time', 'Keywords', 'Campaign',␣
 ↪'Active Page', 'Ad Group',
                                                'Referrer', 'Device Type', 'Browser']]
joined_calls_df_final.columns          = ['Fee', 'Zip', 'City', 'State',␣
 ↪'Source', 'Duration',
                                                'Start Time', 'Keywords','Campaign',␣
 ↪'Page', 'Ad Group',
                                                'Referrer', 'Device Type', 'Browser']
```

[24]:
```
joined_calls_df_final.to_csv('cleaned_calls.csv')
```

3 Census Data

[ ]:
```
census_DP02 = pd.read_csv('ACSDP5Y2019.
 ↪DP02_data_with_overlays_2021-11-10T063909.csv', low_memory=False)
census_DP03 = pd.read_csv('ACSDP5Y2019.
 ↪DP03_data_with_overlays_2021-11-04T190527.csv', low_memory=False)
census_DP04 = pd.read_csv('ACSDP5Y2019.
 ↪DP04_data_with_overlays_2021-11-08T005314.csv', low_memory=False)
census_DP05 = pd.read_csv('ACSDP5Y2019.
 ↪DP05_data_with_overlays_2021-11-04T120142.csv', low_memory=False)
```

[ ]:
```
census_DP02 = census_DP02[['NAME', 'DP02_0001E', 'DP02_0002PE', 'DP02_0003PE',
       'DP02_0006PE', 'DP02_0010PE', 'DP02_0016E', 'DP02_0017E', 'DP02_0026PE',
       'DP02_0027PE', 'DP02_0030PE', 'DP02_0032PE', 'DP02_0033PE',
       'DP02_0036PE', 'DP02_0062PE', 'DP02_0064PE', 'DP02_0065PE',
       'DP02_0066PE', 'DP02_0072PE']]
census_DP03 = census_DP03[['NAME', 'DP03_0001E', 'DP03_0002PE', 'DP03_0009PE',
       'DP03_0047PE', 'DP03_0048PE', 'DP03_0049PE', 'DP03_0062E', 'DP03_0063E',
       'DP03_0088E']]
census_DP04 = census_DP04[['NAME', 'DP04_0041PE', 'DP04_0042PE', 'DP04_0043PE',
       'DP04_0089E', 'DP04_0101E', 'DP04_0110E', 'DP04_0111PE', 'DP04_0112PE',
       'DP04_0113PE', 'DP04_0114PE', 'DP04_0115PE', 'DP04_0134E', 'DP04_0136E',
       'DP04_0137PE', 'DP04_0138PE', 'DP04_0139PE', 'DP04_0140PE',
       'DP04_0141PE', 'DP04_0142PE']]
census_DP05 = census_DP05[['NAME', 'DP05_0001E', 'DP05_0002PE', 'DP05_0003PE',
       'DP05_0018E', 'DP05_0019PE', 'DP05_0023PE', 'DP05_0024PE',
       'DP05_0037PE', 'DP05_0038PE', 'DP05_0044PE', 'DP05_0071PE']]
```

[ ]:
```
census_DP02.columns = ['ZCTA', 'Total Households', 'Percent Married Couple␣
 ↪Family',
                        'Percent Married Couple Family with Children', 'Percent␣
 ↪Male Householder',
                       'Percent Female Householder', 'Average Household Size',␣
 ↪'Average Family Size',
```

```
                      'Percent Males Never Married', 'Percent Males Married',␣
↪'Percent Males Divorced',
                       'Percent Females Never Married','Percent Females␣
↪Married','Percent Females Divorced',
                      'Percent High School Grad', 'Percent Assoc Deg', 'Percent␣
↪Bachelors Deg',
                       'Percent Graduate Deg', 'Percent Disabled']
census_DP03.columns = ['ZCTA', 'Total Pop 16 and Up', 'Percent in Labor Force',␣
↪'Unemployment Rate',
                       'Percent Private Sector', 'Percent Govt Workers',␣
↪'Percent Self Employed',
                       'Median Income', 'Mean Income', 'Per Capita Income']
census_DP04.columns = ['ZCTA', 'Percent 2 Bedroom Homes', 'Percent 3 Bedroom␣
↪Homes', 'Percent 4 Bedroom Homes',
                       'Median House Value', 'Median Mortgage', 'Tot Housing␣
↪Units with Mortgage',
                       'Mortgage Less than 20 Percent of Income', 'Mortgage␣
↪Between 20 and 25 Percent of Income',
                      'Mortgage Between 25 and 30 Percent of Income', 'Mortgage␣
↪Between 30 and 35 Percent of Income',
                      'Mortgage More than 35 Percent of Income', 'Total Units␣
↪Paying Rent',
                       'Rent Less than 15 Percent of Income', 'Rent Between 15␣
↪and 20 Percent of Income',
                       'Rent Between 20 and 25 Percent of Income', 'Rent␣
↪Between 20 and 25 Percent of Income',
                       'Rent Between 25 and 30 Percent of Income', 'Rent␣
↪Between 30 and 35 Percent of Income',
                       'Rent More than 35 Percent of Income']
census_DP05.columns = ['ZCTA', 'Total Pop', 'Percent Male', 'Percent Female',␣
↪'Median Age', 'Percent Under 18',
                       'Percent 62 and Over', 'Percent 65 and Over', 'Percent␣
↪White', 'Percent Black', 'Percent Asian',
                      'Percent Hispanic']
```

[ ]:
```
#join tables
census_df = census_DP02.merge(census_DP03, on='ZCTA').merge(census_DP04,␣
↪on='ZCTA').merge(census_DP05, on='ZCTA')
```

[ ]:
```
#remove NA (some tables do not include Puerto Rico data)
census_df = census_df.dropna().drop(0, axis=0).reset_index(drop=True)
```

[ ]:
```
#remove 'ZCTA5' from each ZCTA
census_df['ZCTA'] = census_df['ZCTA'].str.replace('ZCTA5','')
```

[ ]:
```
census_df
```

[ ]:
```
census_df.to_csv('cleaned_census.csv')
```

[ ]:
```

```

[ ]:
```

```

[ ]:
```

```

[ ]:
```

```

Part 2: Unsupervised Learning Analysis

Code for PCA and K-Means Customer Segmentation

Data Load and Preparation

In [1]:

```
import pandas as pd
pd.set_option('display.max_rows', 1000)
```

In [2]:

```
import seaborn as sns
```

In [3]:

```
#import sales csv
sales = pd.read_csv('cleaned_sales.csv')
census = pd.read_csv('cleaned_census.csv')
```

In [4]:

```
census = census.drop(['Unnamed: 0'], axis=1)
census_subset = census[['ZCTA','Average Household Size', 'Median Age',
'Percent Bachelors Deg' , 'Percent Graduate Deg', 'Median Mortgage', 'Mean
Income', 'Percent 65 and Over', 'Percent White']]
census_subset['ZCTA'] = census_subset['ZCTA'].astype(str)
```

```
C:\Users\julie\anaconda3\lib\site-packages\ipykernel_launcher.py:3:
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_guide/indexing.html#returning-a-view-versus-a-copy
  This is separate from the ipykernel package so we can avoid doing imports
until
```

In [5]:

```
sales['Practice Area'].value_counts()
```

Out[5]:

```
Estate Planning                207
Estate Administration           42
Estate Admin - Package          24
Estate Admin - Hourly           19
Business                        14
Medicaid                        11
Business - Package               7
Estate Admin - Partial           6
Real Estate                      6
Business - LLC                   5
Guardianship                     4
Trust Administration             3
Estate Admin - SEP               3
Business - Hourly                3
Non Profit                       2
Medicaid - Hourly                2
```

```
Real Estate - Deed                    1
POA Agent Rep                         1
Estate Administration - Hourly        1
Medicaid - Package                    1
Name: Practice Area, dtype: int64
```

In [6]:

```
sales.loc[sales['Practice Area'].str.contains('Estate Admin'), 'Practice
Area'] = 'Estate Admin'
sales.loc[sales['Practice Area'].str.contains('Business'), 'Practice Area'] =
'Business'
sales.loc[sales['Practice Area'].str.contains('Medicaid'), 'Practice Area'] =
'Medicaid'
sales.loc[(sales['Practice Area'] != 'Estate Admin') &
           (sales['Practice Area'] !='Estate Planning')  &
          (sales['Practice Area'] !='Business')  &
           (sales['Practice Area']!='Medicaid'), 'Practice Area'] = 'Other'

sales['Practice Area'].value_counts()
```

Out[6]:

```
Estate Planning    207
Estate Admin        95
Business            29
Other               17
Medicaid            14
Name: Practice Area, dtype: int64
```

In [7]:

```
#join on zip
joined = sales.merge(census_subset, left_on='Zip', right_on='ZCTA')

#create column for bachelors and graduate degrees
joined['Degree'] = joined['Percent Bachelors Deg'].astype(float) +
joined['Percent Graduate Deg'].astype(float)

#one-hot encoding of practice area
dummies = pd.get_dummies(joined['Practice Area'])

#remove unneeded columns
joined = joined.drop(['Unnamed: 0', 'Percent Bachelors Deg',  'Percent
Graduate Deg',
                      'City', 'State', 'Zip', 'ZCTA', 'Contact Year',
'Practice Area', 'Referral'],axis=1)
joined['Median Mortgage'] = joined['Median
Mortgage'].str.replace('+','').str.replace(',','')

#concat all columns and remove rows with no fee
joined = pd.concat([joined,dummies],axis=1).dropna().reset_index(drop=True)
```

```
C:\Users\julie\anaconda3\lib\site-packages\ipykernel_launcher.py:13:
FutureWarning: The default value of regex will change from True to False in a
future version. In addition, single character regular expressions will*not*
be treated as literal strings when regex=True.
  del sys.path[0]
```

In [8]:

```
numeric = joined[['Fee', 'Average Household Size', 'Median Age',
       'Median Mortgage', 'Mean Income', 'Percent 65 and Over',
       'Percent White', 'Degree']].apply(pd.to_numeric)
categoric = joined[['Contact Month','Business', 'Estate Admin',
       'Estate Planning','Medicaid', 'Other']]

df = pd.concat([categoric, numeric], axis=1)
```

In [9]:

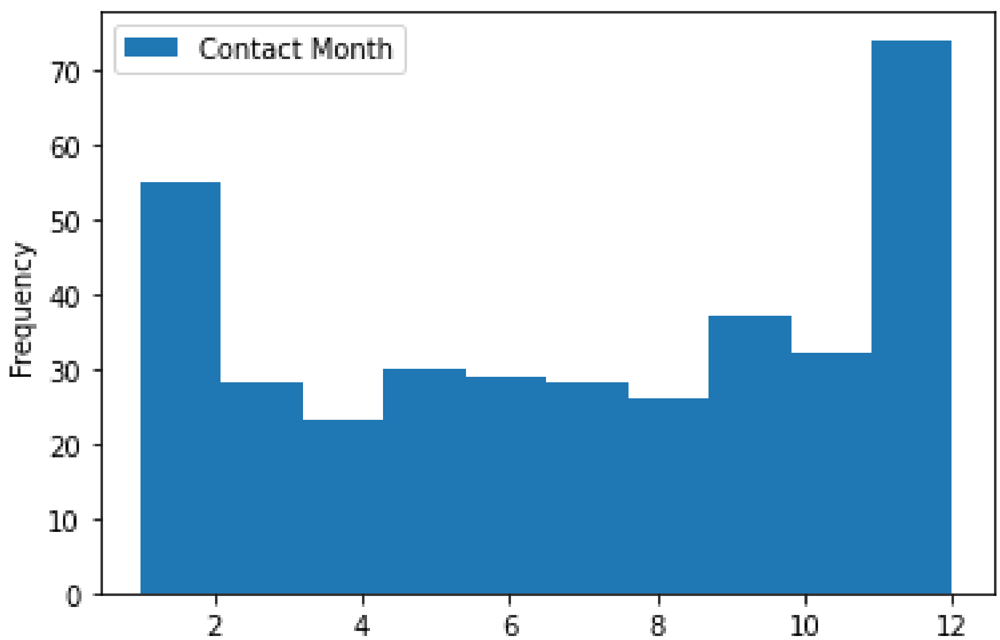
```
#df[['Fee', 'Average Household Size', 'Median Age',
#        'Median Mortgage', 'Mean Income', 'Percent 65 and Over',
#        'Percent White', 'Degree']].describe()
```

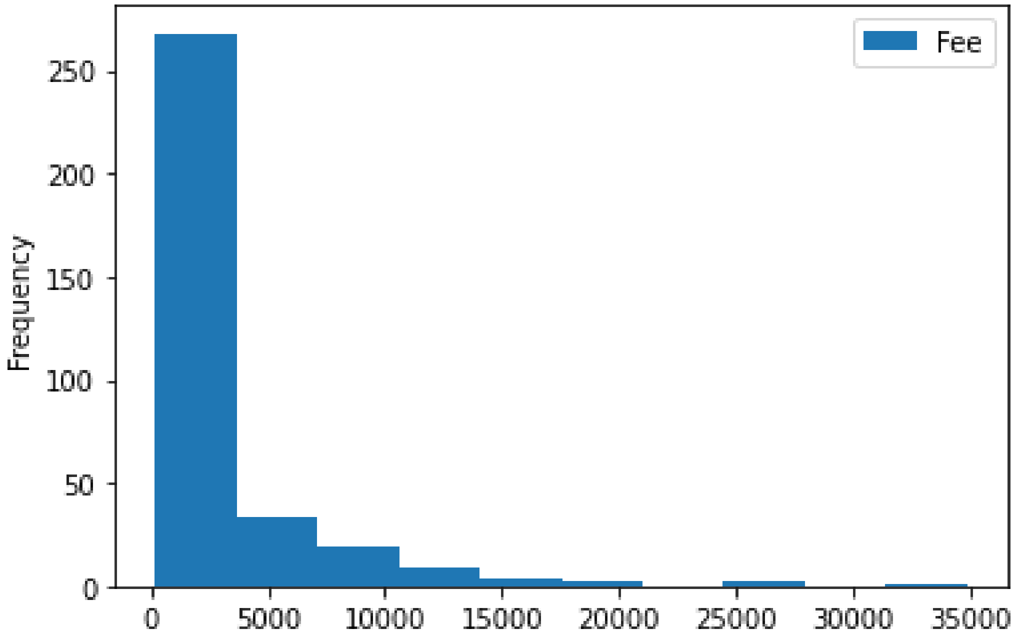
In [10]:

```
import matplotlib.pyplot as plt

ax = sales[['Contact Month']].plot.hist()
plt.show()

ax = sales[['Fee']].plot.hist()
plt.show()
```



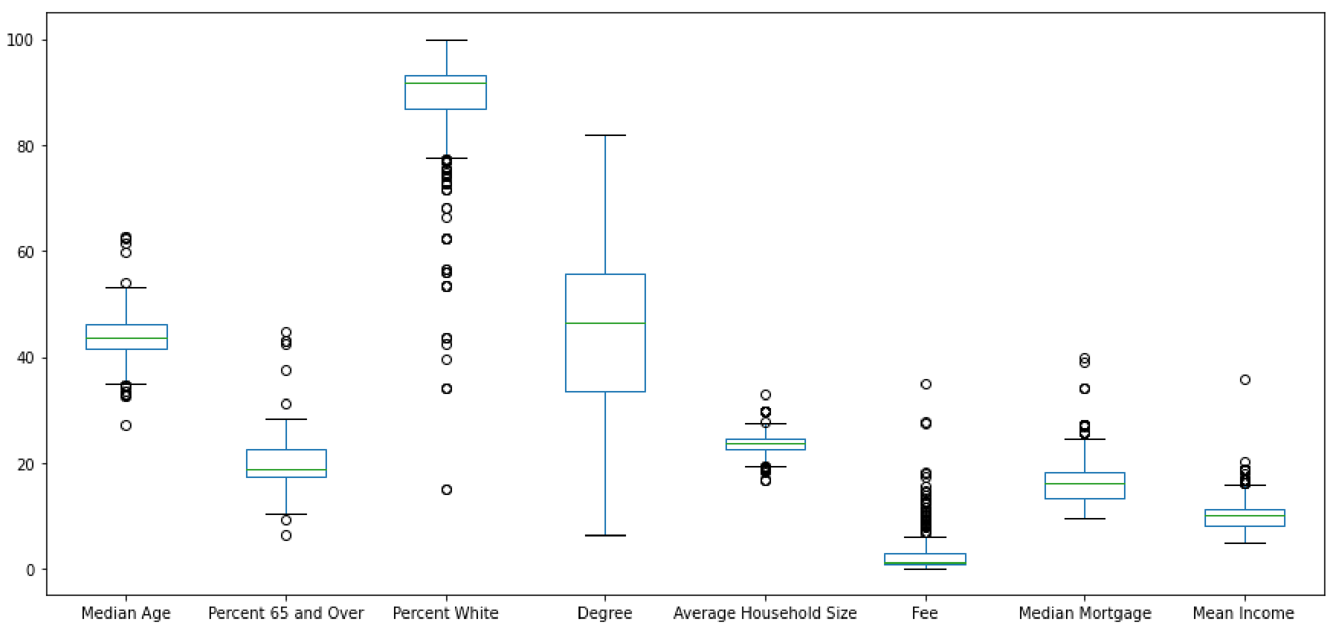


In [11]:

```
plotting_df = df.copy()
plotting_df['Mean Income'] = df['Mean Income']/10000
plotting_df['Fee'] = df['Fee']/1000
plotting_df['Median Mortgage'] = df['Median Mortgage']/100
plotting_df['Average Household Size'] = df['Average Household Size']*10
plotting_df[['Median Age','Percent 65 and Over',
       'Percent White', 'Degree','Average Household Size','Fee', 'Median
Mortgage', 'Mean Income']].plot.box(figsize=(15,7))
```

Out[11]:

```
<AxesSubplot:>
```



In [12]:

```
from sklearn.preprocessing import StandardScaler

# Instantiate scaler
scaler = StandardScaler()

numeric_scaled = pd.DataFrame(scaler.fit_transform(numeric),
columns=numeric.columns)

# Concatenate categoric and scaled numeric columns
scaled_DF = pd.concat([categoric, numeric_scaled], axis=1)
```

PCA

In [13]:

```
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
principalComponents = pca.fit_transform(scaled_DF)
principalDf = pd.DataFrame(data = principalComponents
             , columns = ['principal component 1', 'principal component 2'])
```

In [14]:

```
pca.explained_variance_ratio_
```

Out[14]:

```
array([0.59442492, 0.15070333])
```

In [15]:

```
print(scaled_DF.columns)
print(pca.components_)
```

```
Index(['Contact Month', 'Business', 'Estate Admin', 'Estate Planning',
       'Medicaid', 'Other', 'Fee', 'Average Household Size', 'Median Age',
       'Median Mortgage', 'Mean Income', 'Percent 65 and Over',
       'Percent White', 'Degree'],
      dtype='object')
[[-9.98965670e-01 -2.61358640e-03 -1.21751676e-03  2.61921441e-03
   5.85558065e-03 -4.64369189e-03  2.57409583e-03  3.43537820e-03
   2.97625745e-02 -2.88096399e-03 -1.04542425e-02  2.34024155e-02
   1.14688773e-02 -1.71909230e-02]
 [-2.95350112e-02  8.37680195e-03 -4.14276385e-03 -3.96034179e-04
   2.01727963e-03 -5.85528355e-03  9.93355754e-02  3.43991902e-01
  -2.43606623e-01  4.86551934e-01  4.89006048e-01 -3.54962082e-01
  -1.08665448e-01  4.44740046e-01]]
```

K-Means Clustering

In [16]:

```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import numpy as np
```

In [17]:

```
# Import module
from sklearn.cluster import KMeans

# Instantiate
kmeans = KMeans(n_clusters=4, random_state=123)

# Fit
fit = kmeans.fit(scaled_DF)

# Print inertia
print("Sum of squared distances for 4 clusters is", kmeans.inertia_)
```

```
Sum of squared distances for 4 clusters is 2842.8272368509147
```

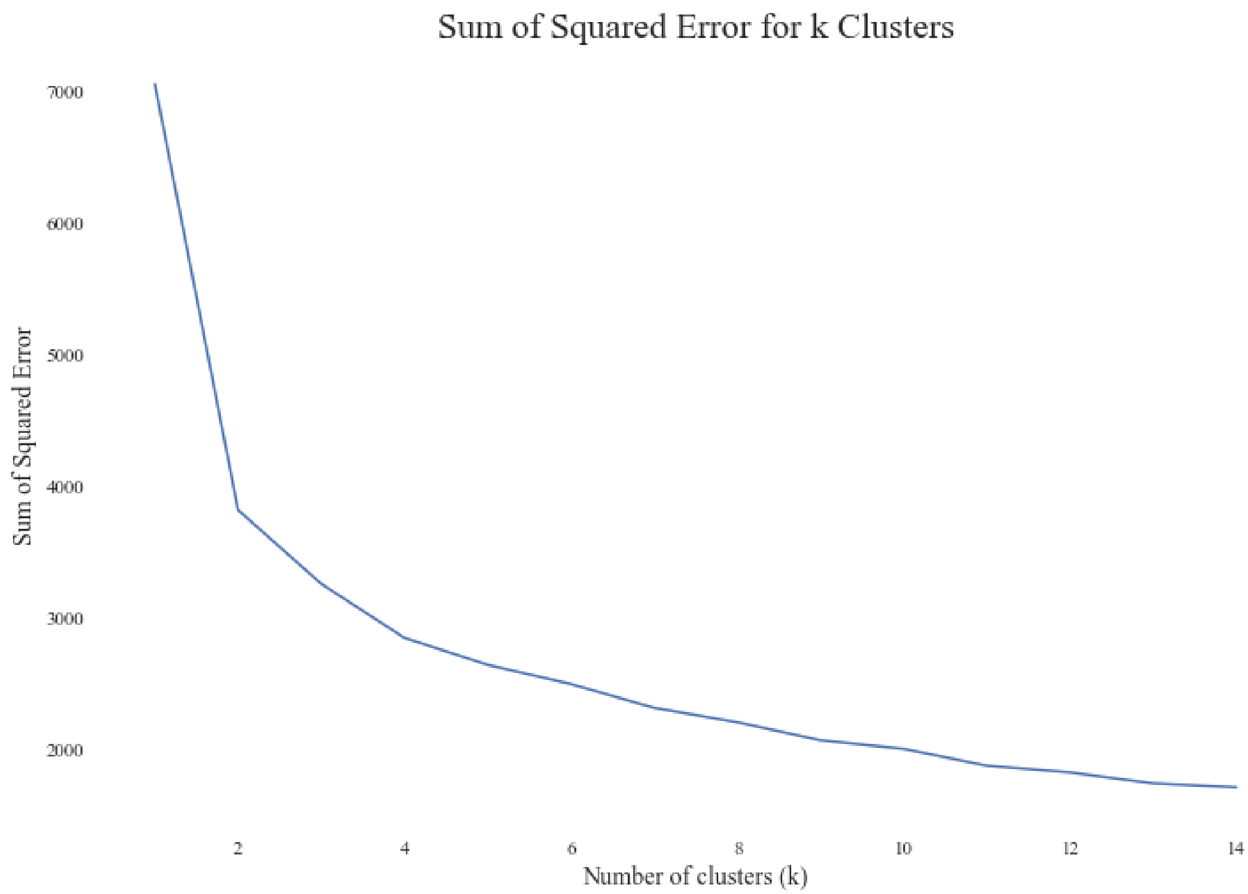
In [18]:

```
cluster_score = []
for k in range(1,15):
    k_means_model = KMeans(n_clusters=k)
    k_means_model.fit(scaled_DF)
    cluster_score.append(k_means_model.inertia_)
```

```
C:\Users\julie\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:882:
UserWarning: KMeans is known to have a memory leak on Windows with MKL, when
there are less chunks than available threads. You can avoid it by setting the
environment variable OMP_NUM_THREADS=2.
  f"KMeans is known to have a memory leak on Windows "
```

In [19]:

```
sns.set_theme(style="white")
sns.set_style({'axes.facecolor':'white', 'font.family':'Times New Roman'})
fig = plt.figure(figsize = (12,8))
ax = fig.add_subplot(1,1,1)
plt.xlabel('Number of clusters (k)', fontsize=14)
plt.ylabel('Sum of Squared Error', fontsize=14)
plt.title('Sum of Squared Error for k Clusters', fontsize=20)
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
ax.spines['bottom'].set_visible(False)
ax.spines['left'].set_visible(False)
plt.plot(range(1,15), cluster_score, '-')
plt.show()
```



In [20]:

```
k_means_model = KMeans(n_clusters=4, random_state=1)
k_means_model.fit(scaled_DF)
scaled_DF['k_means_values']=k_means_model.predict(scaled_DF)
scaled_DF_x = scaled_DF.copy()
scaled_DF_x = scaled_DF_x.drop('k_means_values', axis=1)
```

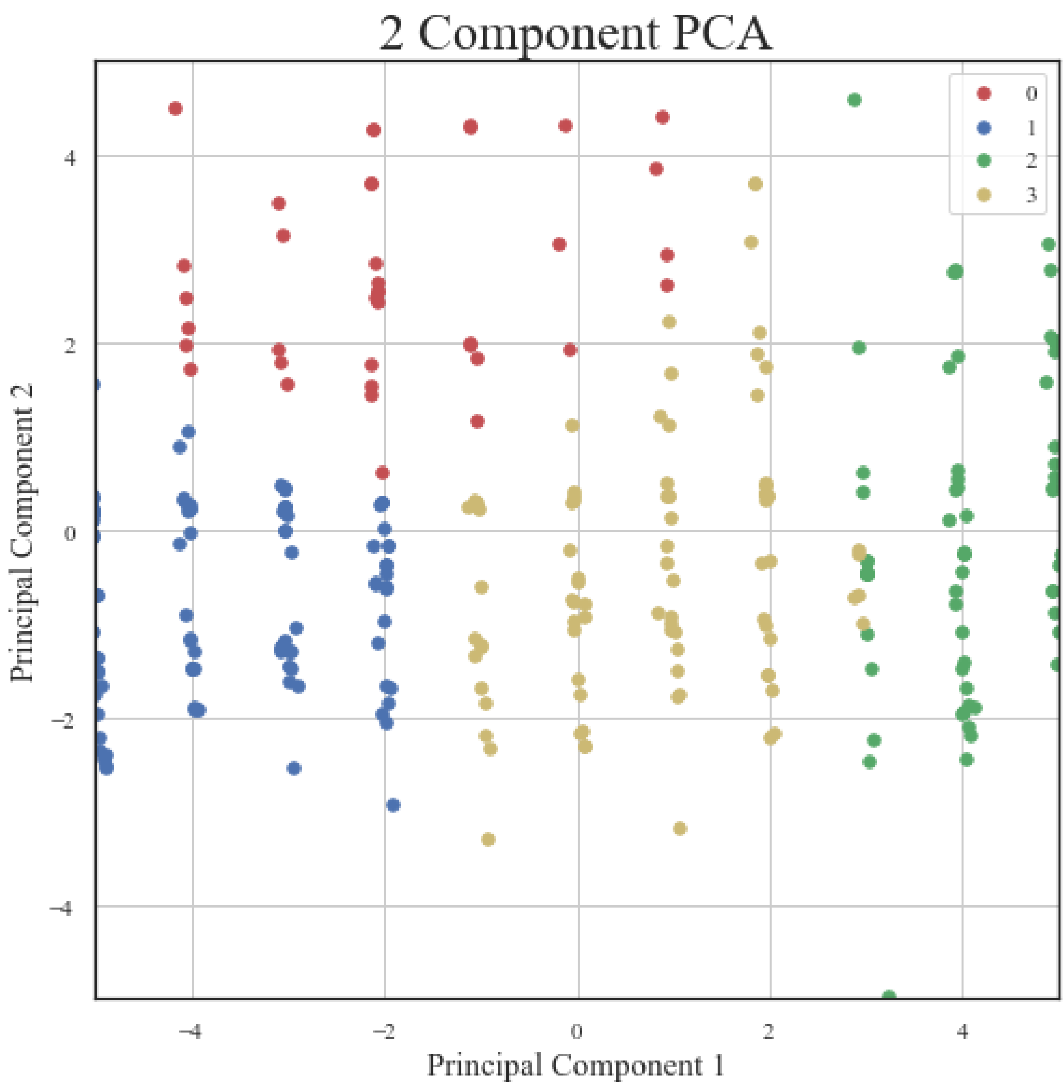
In [21]:

```
finalDf = pd.concat([principalDf, scaled_DF['k_means_values']], axis = 1)
```

In [22]:

```
fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Principal Component 1', fontsize = 15)
ax.set_ylabel('Principal Component 2', fontsize = 15)
ax.set_title('2 Component PCA', fontsize = 24)
plt.xlim([-5, 5])
plt.ylim([-5, 5])
cluster = [0,1,2,3]
color = ['r', 'b', 'g','y']
for cluster, color in zip(cluster,color):
    clusInd = finalDf['k_means_values'] == cluster
    ax.scatter(finalDf.loc[clusInd, 'principal component 1']
               , finalDf.loc[clusInd, 'principal component 2']
```

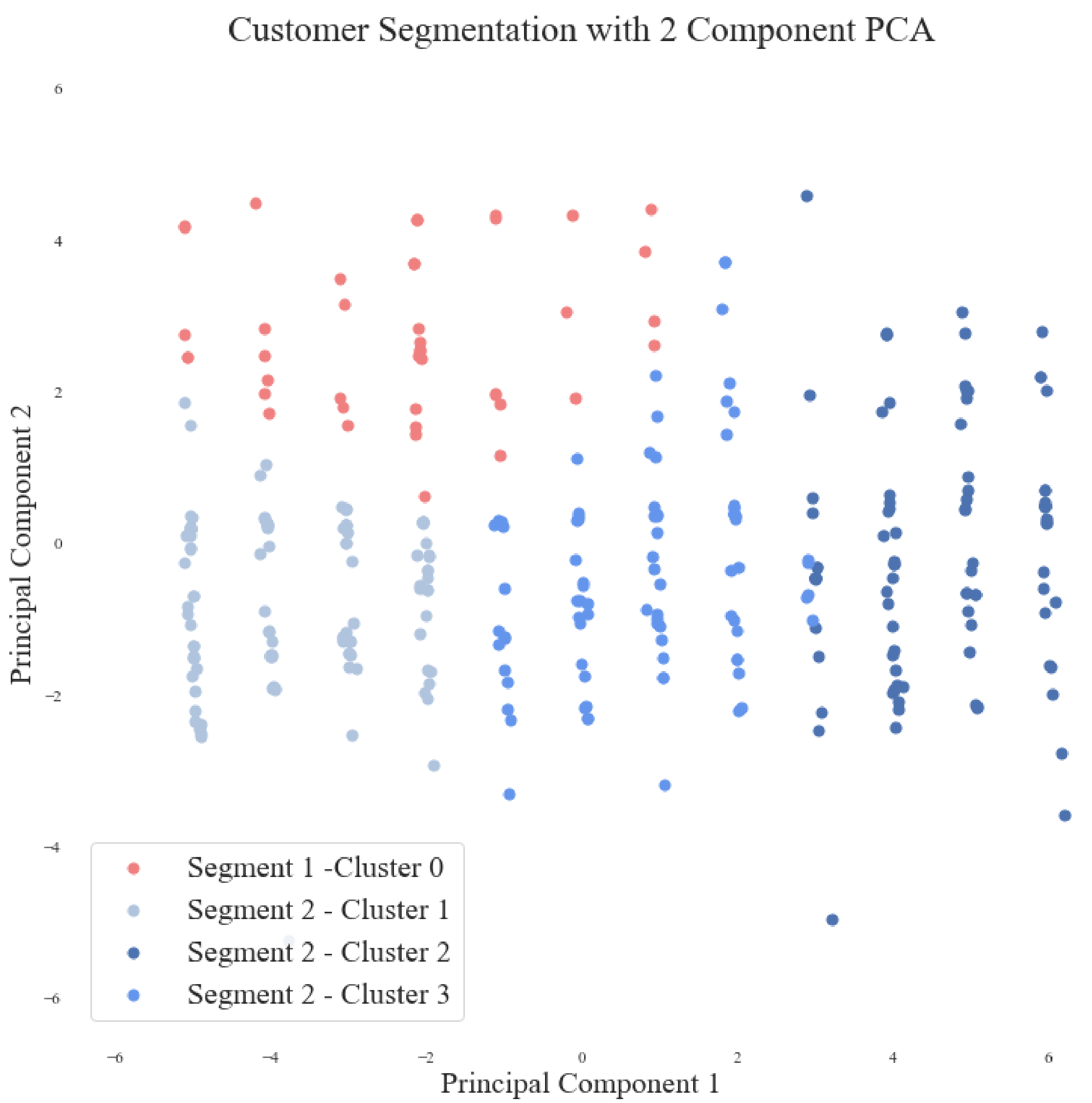
```
                , c = color
                , s = 30)
ax.legend([0,1,2,3])
ax.grid()
plt.show()
```



In [23]:

```
sns.set_theme(style="white")
sns.set_style({'axes.facecolor':'white', 'font.family':'Times New Roman'})
fig = plt.figure(figsize = (12,12))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Principal Component 1', fontsize = 20)
ax.set_ylabel('Principal Component 2', fontsize = 20)
ax.set_title('Customer Segmentation with 2 Component PCA', fontsize = 24)
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
ax.spines['bottom'].set_visible(False)
ax.spines['left'].set_visible(False)
```

```
plt.xlim([-6.5, 6.5])
plt.ylim([-6.5, 6.5])
plt.text(-8, -8, 'PC1 is primarily composed of \'Contact Month\'', ha='left',
fontsize = 16)
plt.text(-8, -8.5, 'PC2 is primarily composed of \'Mean Income\', \'Median
Mortgage\', \'Percent Degree\', \'Average Household Size, and \'Percent 65
and Over\'', ha='left', fontsize = 16)
cluster = [0,1,2,3]
color = ['lightcoral', 'lightsteelblue', 'b', 'cornflowerblue']
for cluster, color in zip(cluster,color):
    clusInd = finalDf['k_means_values'] == cluster
    ax.scatter(finalDf.loc[clusInd, 'principal component 1']
               , finalDf.loc[clusInd, 'principal component 2']
               , c = color
               , s = 50)
ax.legend(['Segment 1 -Cluster 0','Segment 2 - Cluster 1', 'Segment 2 -
Cluster 2', 'Segment 2 - Cluster 3'], fontsize=20, loc = 'lower left')
plt.show()
```



PC1 is primarily composed of 'Contact Month'

PC2 is primarily composed of 'Mean Income', 'Median Mortgage', 'Percent Degree', 'Average Household Size, and 'Percent 65 and Over'

In [24]:

```
df_with_clusters = pd.concat([df, scaled_DF['k_means_values']], axis = 1)
```

In [25]:

```
df_with_clusters[df_with_clusters['k_means_values']==0].describe()
```

Out[25]:

	Contact Month	Business	Estate Admin	Estate Planning	Medicaid	Other	Fee	Average Household Size	Median Age	Median Mortgage	Mean Income	Percent 65 and Over	Percent White	Degree	k_means_values
count	44.000000	44.000000	44.000000	44.000000	44.000000	44.000000	44.000000	44.000000	44.000000	44.000000	44.000000	44.000000	44.000000	44.000000	44.0
mean	9.159091	0.113636	0.340909	0.500000	0.022727	0.022727	5609.625000	2.592273	41.290909	2416.840909	159112.840909	15.697727	84.377273	63.729545	0.0
std	1.724558	0.321038	0.479495	0.505781	0.150756	0.150756	7888.049783	0.313538	3.850691	507.350892	40947.797211	3.932304	13.687048	9.346970	0.0
min	6.000000	0.000000	0.000000	0.000000	0.000000	0.000000	250.000000	1.690000	32.900000	1552.000000	85565.000000	6.300000	42.400000	33.300000	0.0
25%	8.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1000.000000	2.440000	40.100000	2083.000000	135569.750000	11.700000	80.150000	59.000000	0.0
50%	9.000000	0.000000	0.000000	0.500000	0.000000	0.000000	1800.000000	2.630000	41.900000	2271.000000	166720.000000	16.550000	89.800000	66.150000	0.0
75%	10.250000	0.000000	1.000000	1.000000	0.000000	0.000000	6500.000000	2.740000	43.700000	2706.000000	171881.500000	18.800000	93.100000	68.525000	0.0
max	12.000000	1.000000	1.000000	1.000000	1.000000	1.000000	34885.000000	3.290000	49.400000	4000.000000	358261.000000	24.900000	95.900000	82.000000	0.0

In [26]:

```
cluster1 =
df_with_clusters[df_with_clusters['k_means_values']==0].describe().loc['mean'
,:]
```

In [27]:

```
cluster2 =
df_with_clusters[df_with_clusters['k_means_values']==1].describe().loc['mean'
,:]
```

In [28]:

```
cluster3 =
df_with_clusters[df_with_clusters['k_means_values']==2].describe().loc['mean'
,:]
```

In [29]:

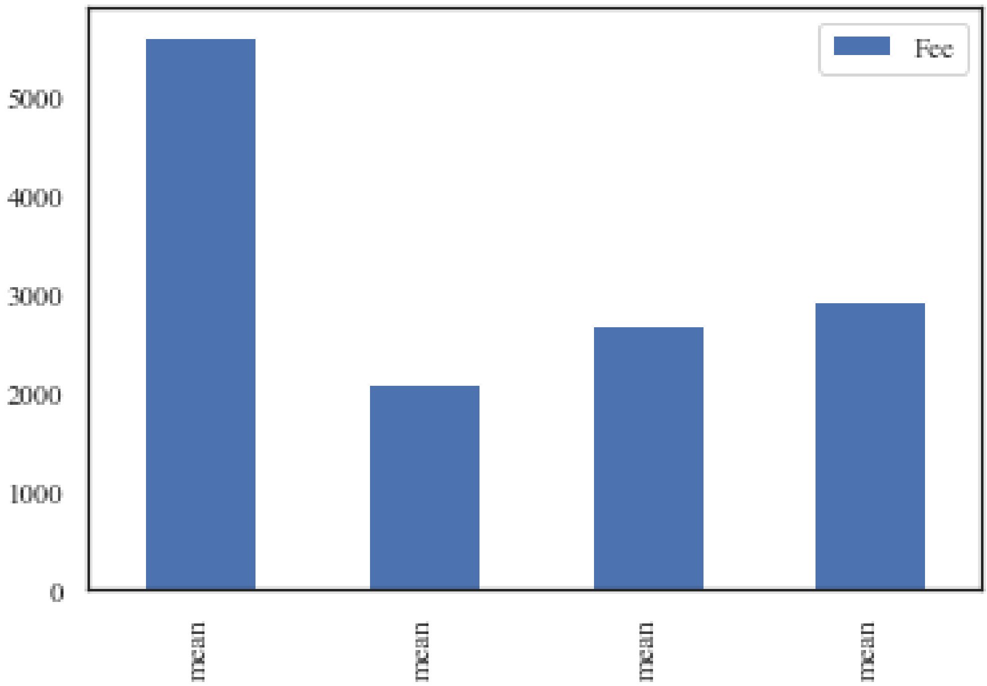
```
cluster4 =
df_with_clusters[df_with_clusters['k_means_values']==3].describe().loc['mean'
,:]
```

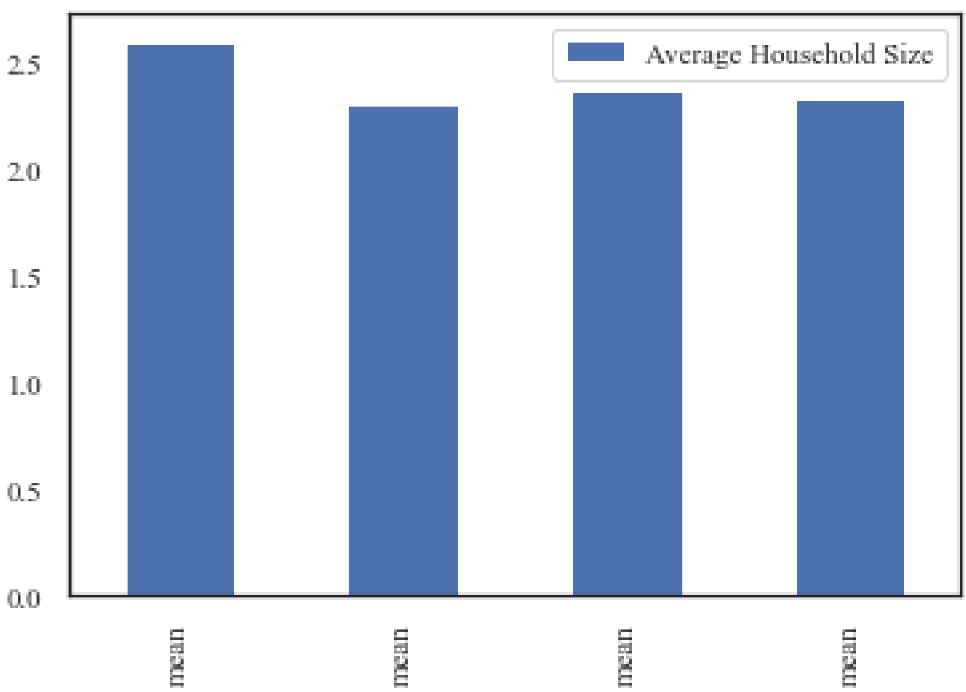
In [30]:

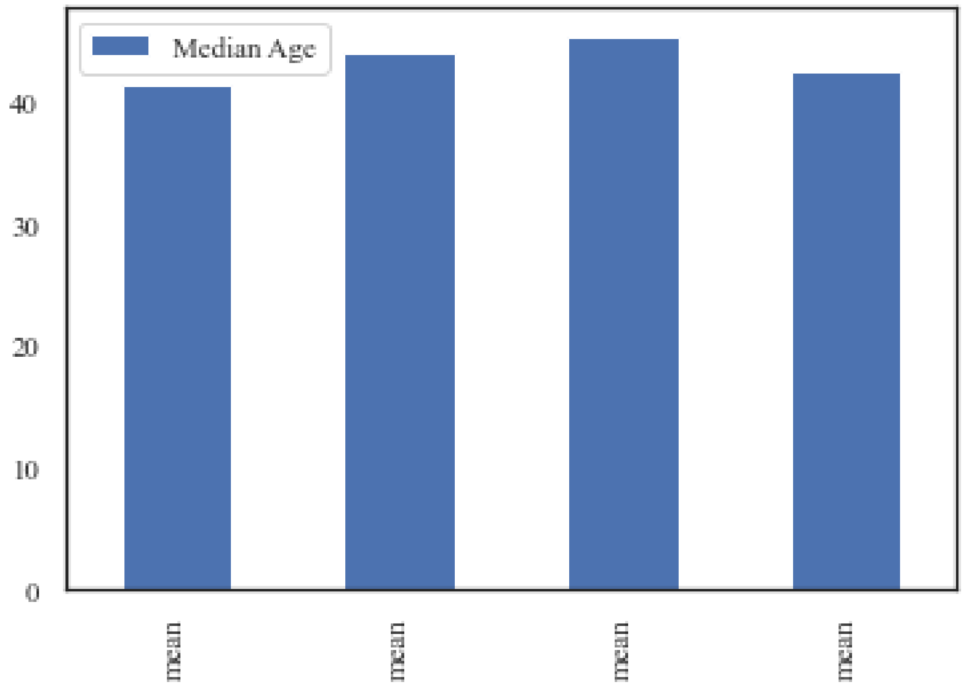
```
list_of_series = [cluster1,cluster2,cluster3,cluster4]
cluster_df = pd.DataFrame(list_of_series)
cluster_df[['Fee']].plot.bar()
cluster_df[['Average Household Size']].plot.bar()
cluster_df[['Median Age']].plot.bar()
cluster_df[['Median Mortgage']].plot.bar()
cluster_df[['Mean Income']].plot.bar()
cluster_df[['Percent 65 and Over']].plot.bar()
cluster_df[['Percent White']].plot.bar()
cluster_df[['Degree']].plot.bar()
cluster_df[['Contact Month']].plot.bar()
```

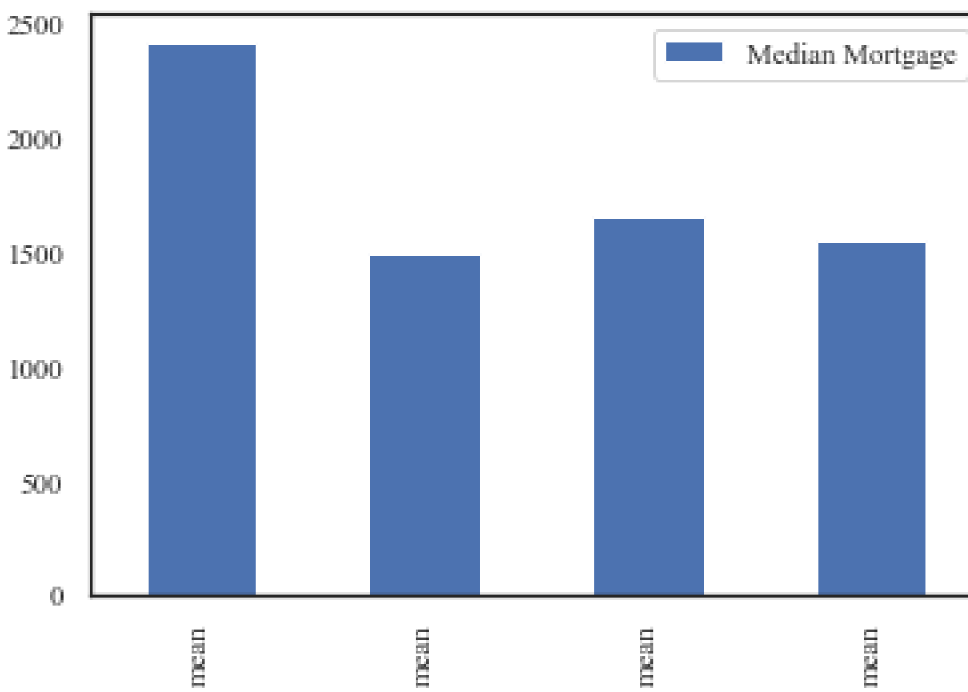
Out[30]:

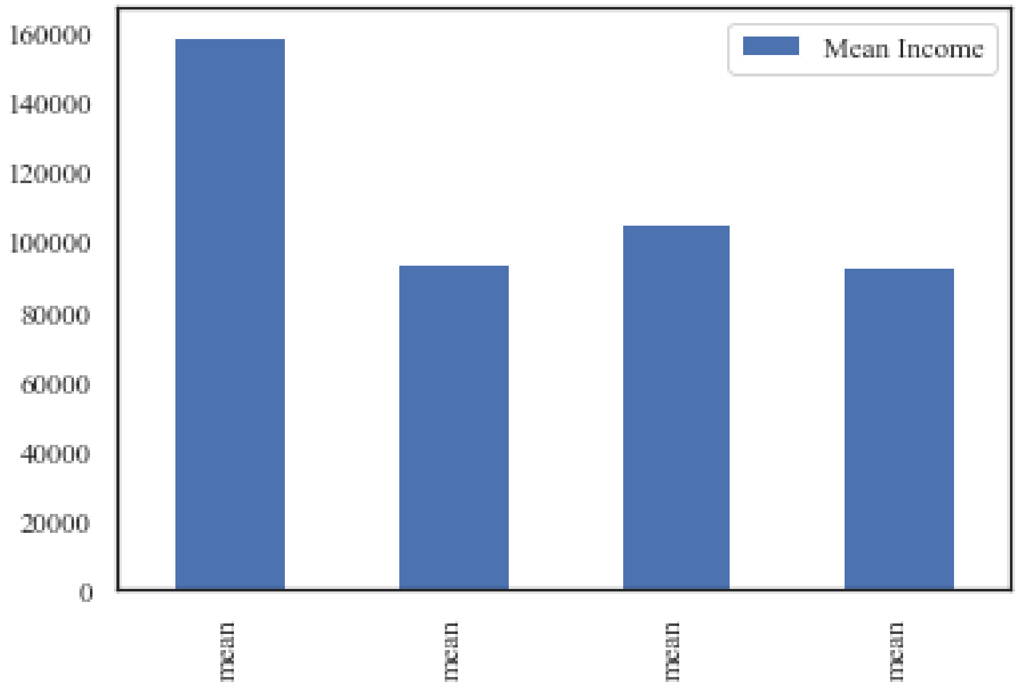
```
<AxesSubplot:>
```

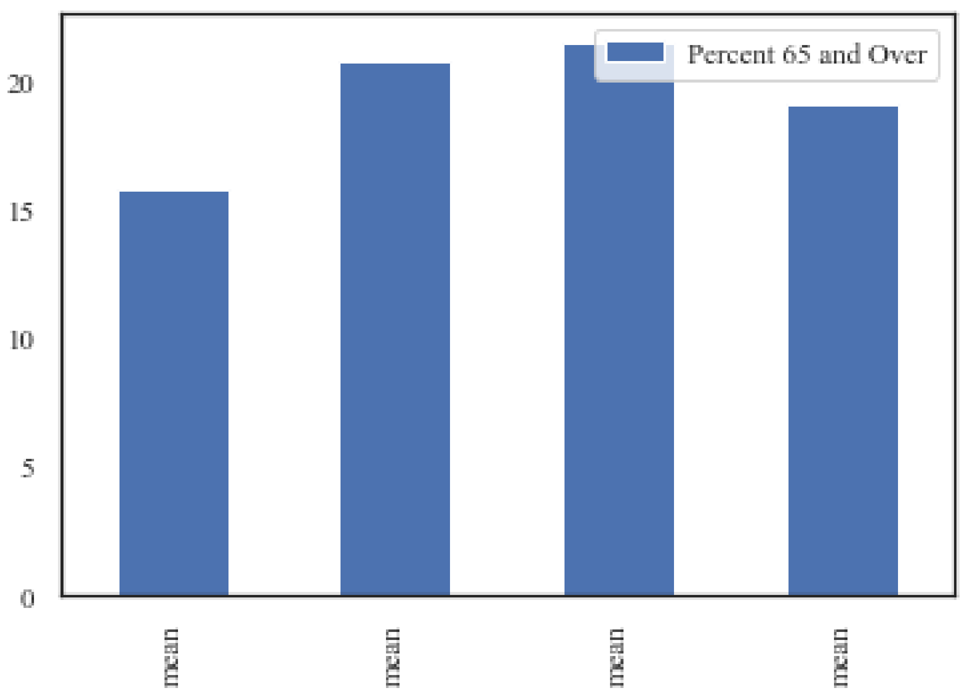

Fee
5000
4000
3000
2000
1000
0
mean
mean
mean
mean

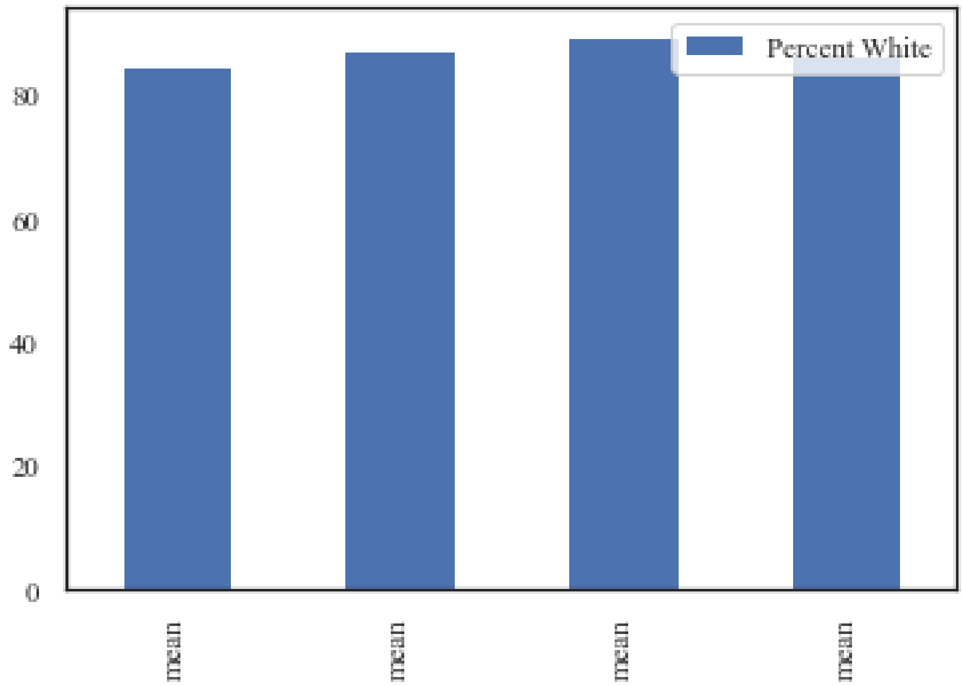

Average Household Size
2.5
2.0
1.5
1.0
0.5
0.0
mean
mean
mean
mean

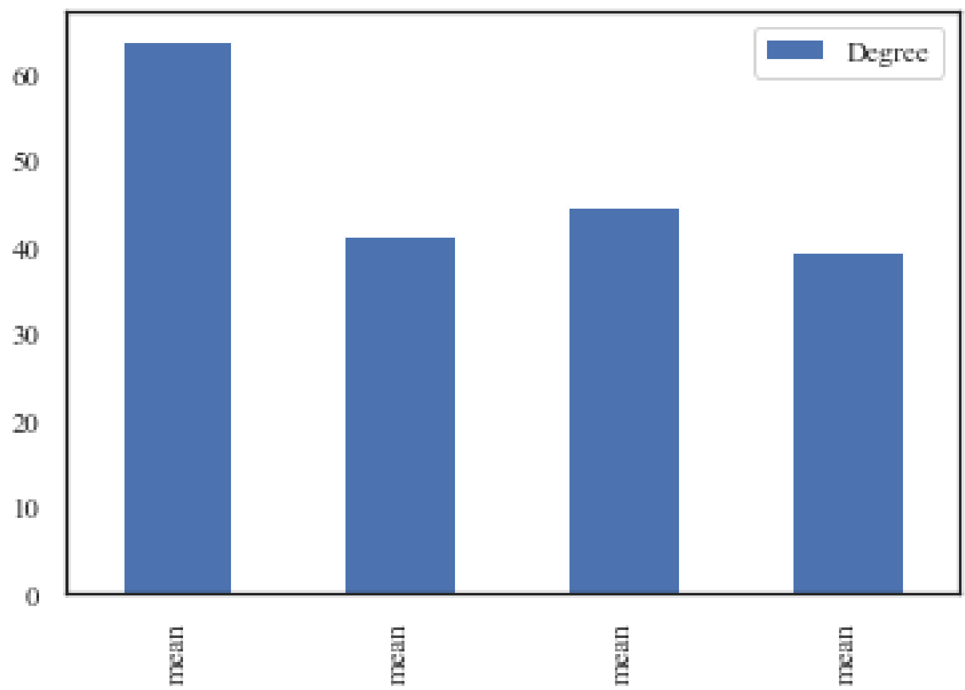

Median Age
0
10
20
30
40
mean
mean
mean
mean

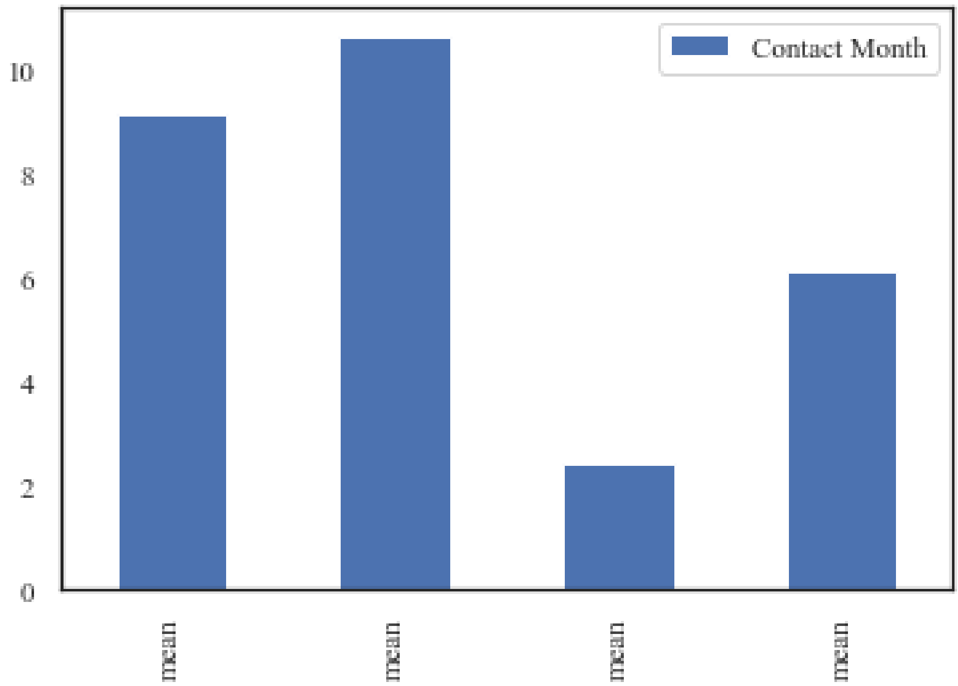

Median Mortgage
0
500
1000
1500
2000
2500
mean
mean
mean
mean


Mean Income
160000
140000
120000
100000
80000
60000
40000
20000
0
mean
mean
mean
mean


Percent 65 and Over
20
15
10
5
0
mean
mean
mean
mean


Percent White
0
20
40
60
80
mean
mean
mean
mean


Degree
0
10
20
30
40
50
60
mean
mean
mean
mean



In [31]:

```
cluster_df
```

Out[31]:

	Contact Month	Business	Estate Admin	Estate Planning	Medicaid	Other	Fee	Average Household Size	Median Age	Median Mortgage	Mean Income	Percent 65 and Over	Percent White	Degree	k_means_values
mean	9.159091	0.113636	0.340909	0.500000	0.022727	0.022727	5609.625000	2.592273	41.290909	2416.840909	159112.840909	15.697727	84.377273	63.729545	0.0
mean	10.660550	0.082569	0.211009	0.623853	0.009174	0.073394	2073.577982	2.295138	43.905505	1490.532110	93246.724771	20.657798	87.100917	41.125688	1.0
mean	2.408602	0.075269	0.215054	0.634409	0.043011	0.032258	2681.236559	2.356344	45.386022	1659.139785	105114.688172	21.476344	89.393548	44.556989	2.0
mean	6.125000	0.034091	0.340909	0.488636	0.079545	0.056818	2910.971591	2.324659	42.460227	1544.215909	92775.931818	19.002273	86.215909	39.238636	3.0

In [32]:

```
cluster1_areas =
df_with_clusters[df_with_clusters['k_means_values']==0][['Estate Planning',
'Estate Admin', 'Medicaid',
'Business','Other']].sum()/df_with_clusters[df_with_clusters['k_means_values'
]==0].shape[0]
cluster1_areas
```

Out[32]:

```
Estate Planning    0.500000
Estate Admin       0.340909
Medicaid           0.022727
Business           0.113636
Other              0.022727
dtype: float64
```

In [33]:

```
cluster2_areas =
df_with_clusters[df_with_clusters['k_means_values']==1][['Estate Planning',
'Estate Admin', 'Medicaid',
'Business','Other']].sum()/df_with_clusters[df_with_clusters['k_means_values'
]==1].shape[0]
```

```
cluster2_areas
```

Out[33]:

```
Estate Planning    0.623853
Estate Admin       0.211009
Medicaid           0.009174
Business           0.082569
Other              0.073394
dtype: float64
```

In [34]:

```
cluster3_areas =
df_with_clusters[df_with_clusters['k_means_values']==2][['Estate Planning',
'Estate Admin', 'Medicaid',
'Business','Other']].sum()/df_with_clusters[df_with_clusters['k_means_values'
]==2].shape[0]
cluster3_areas
```

Out[34]:

```
Estate Planning    0.634409
Estate Admin       0.215054
Medicaid           0.043011
Business           0.075269
Other              0.032258
dtype: float64
```

In [35]:

```
cluster4_areas =
df_with_clusters[df_with_clusters['k_means_values']==3][['Estate Planning',
'Estate Admin', 'Medicaid',
'Business','Other']].sum()/df_with_clusters[df_with_clusters['k_means_values'
]==3].shape[0]
cluster4_areas
```

Out[35]:

```
Estate Planning    0.488636
Estate Admin       0.340909
Medicaid           0.079545
Business           0.034091
Other              0.056818
dtype: float64
```

In [ ]:

Part 3: Supervised Learning Analysis

Code for call log classification with logistic regression and random forest

In [1]:

```
import pandas as pd
pd.set_option('display.max_rows', 1000)

import numpy as np
from numpy import mean

import matplotlib.pyplot as plt

#turn off warnings for final run
import warnings
warnings.filterwarnings('ignore')
```

In [2]:

```
from sklearn import metrics
from sklearn.preprocessing import StandardScaler

from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV

from sklearn.feature_selection import RFE
from sklearn.feature_selection import RFECV

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import RandomForestClassifier


from statsmodels.stats.outliers_influence import variance_inflation_factor
```

Data Load and Preparation

In [3]:

```
#import sales csv
calls = pd.read_csv('cleaned_calls.csv')
```

In [4]:

```
#ceate column to indicate if call ended in sale based on customer existance
calls['sale'] = 0
calls.loc[calls['Zip'].notnull(), ['sale']] = 1
```

In [5]:

```
#only include a subset of the variables
```

```
calls_subset = calls[['Duration', 'Start Time', 'Keywords', 'Campaign', 'Page',
                      'Device Type', 'Browser', 'Referrer', 'sale', 'State']]
calls_subset
```

Out[5]:

	Duration	Start Time	Keywords	Campaign	Page	Device Type	Browser	Referrer	sale	State
0	167	12/22/2021 12:45	NaN	NaN	main/estate-planning-ppc/	mobile	Chrome	www.google.com	0	PA
1	115	12/22/2021 12:36	NaN	Medicaid - Elder Law - 001	NaN	NaN	NaN	NaN	0	IA
2	106	12/22/2021 12:23	NaN	NaN	main/	desktop	Chrome	direct	0	PA
3	33	12/22/2021 12:01	NaN	NaN	NaN	NaN	NaN	NaN	0	PA
4	112	12/22/2021 11:47	NaN	NaN	NaN	NaN	NaN	NaN	0	CA
...	...	...	...	...	...	...	...	...	...	...
1780	18	1/5/2021 8:47	NaN	NaN	NaN	NaN	NaN	NaN	0	PA
1781	44	1/5/2021 8:23	power of attorney will	Estate Planning 005	main/estate-planning-ppc/	desktop	Chrome	www.google.com	0	NY
1782	188	1/4/2021 14:28	NaN	NaN	NaN	NaN	NaN	NaN	1	PA
1783	91	1/4/2021 13:12	NaN	NaN	main/contact/	desktop	Chrome	www.google.com	0	MO
1784	26	1/4/2021 8:45	NaN	NaN	NaN	NaN	NaN	NaN	0	PA

1785 rows × 10 columns

In [6]:

```
#convert start time to month and hour variables
calls_subset['Month'] = calls_subset['Start Time'].apply(lambda x: x.split(' ')[0].split('/')[0])
calls_subset['Hour'] = calls_subset['Start Time'].apply(lambda x: x.split(' ')[1].split(':')[0])
calls_subset = calls_subset.drop('Start Time', axis=1)
calls_subset
```

Out[6]:

	Duration	Keywords	Campaign	Page	Device Type	Browser	Referrer	sale	State	Month	Hour
0	167	NaN	NaN	main/estate-planning-ppc/	mobile	Chrome	www.google.com	0	PA	12	12
1	115	NaN	Medicaid - Elder Law - 001	NaN	NaN	NaN	NaN	0	IA	12	12
2	106	NaN	NaN	main/	desktop	Chrome	direct	0	PA	12	12

	Duration	Keywords	Campaign	Page	Device Type	Browser	Referrer	sale	State	Month	Hour
3	33	NaN	NaN	NaN	NaN	NaN	NaN	0	PA	12	12
4	112	NaN	NaN	NaN	NaN	NaN	NaN	0	CA	12	11
...	...	...	...	...	...	...	...	...	...	...	...
1780	18	NaN	NaN	NaN	NaN	NaN	NaN	0	PA	1	8
1781	44	power of attorney will	Estate Planning 005	main/estate-planning-ppc/	desktop	Chrome	www.google.com	0	NY	1	8
1782	188	NaN	NaN	NaN	NaN	NaN	NaN	1	PA	1	14
1783	91	NaN	NaN	main/contact/	desktop	Chrome	www.google.com	0	MO	1	13
1784	26	NaN	NaN	NaN	NaN	NaN	NaN	0	PA	1	8

1785 rows × 11 columns

In [7]:

```
#remove rows without full data
calls_noNa = calls_subset.dropna().reset_index(drop=True)
print('Number of records: ' + str(calls_noNa.shape[0]))
print('Number of records with sale: ' + str(calls_noNa[calls_noNa['sale'] ==
1].reset_index().shape[0]))
Number of records: 401
Number of records with sale: 48
```

Get Dummies

In [8]:

```
#get raw value counts for variables
#for i in ['Keywords', 'Campaign', 'Page', 'Device Type', 'Browser',
'Referrer']:
#    print(calls_noNa[i].value_counts())
```

In [9]:

```
#update keyword categories to include at least 15 per category
keyword_categories = ['probate and estate administration lawyers',
'medicaid planning attorney',
'pittsburgh probate lawyers',
'elder law attorney',
'estate administration lawyer near me',
'pennsylvania lawyers']
calls_noNa['Keywords Categories'] = calls_noNa['Keywords']
calls_noNa.loc[~calls_noNa['Keywords'].isin(keyword_categories), 'Keywords
Categories'] = 'Other Keywords'
calls_noNa = calls_noNa.drop('Keywords', axis=1)
calls_noNa['Keywords Categories'].value_counts()
```

Out[9]:

```
Other Keywords                          288
```

```
probate and estate administration lawyers     25
medicaid planning attorney                    22
pittsburgh probate lawyers                    20
elder law attorney                            16
estate administration lawyer near me          15
pennsylvania lawyers                          15
Name: Keywords Categories, dtype: int64
```

In [10]:

```
#update page categories to include at least 15 per category
page_categories = ['elder-law-ppc/',
'estate-planning-ppc/',
'estate-admin-ppc/',
'main/',
'main/pittsburgh-estate-planning-lawyer/power-of-attorney-pittsburgh-pa/',
'main/firstname-redacted/',
'main/contact/']

calls_noNa['Page Categories'] = calls_noNa['Page']
calls_noNa.loc[~calls_noNa['Page'].isin(page_categories), 'Page Categories']
= 'Other Page'
calls_noNa = calls_noNa.drop('Page', axis=1)
calls_noNa['Page Categories'].value_counts()
```

Out[10]:

```
Other Page
322
main/
28
main/pittsburgh-estate-planning-lawyer/power-of-attorney-pittsburgh-pa/
19
main/contact/
16
main/firstname-redacted/
16
Name: Page Categories, dtype: int64
```

In [11]:

```
#update browser categories to include at least 10 per category
browser_categories = ['Chrome', 'Safari']
calls_noNa['Browser Categories'] = calls_noNa['Browser']
calls_noNa.loc[~calls_noNa['Browser'].isin(browser_categories), 'Browser
Categories'] = 'Firefox or Opera'
calls_noNa = calls_noNa.drop('Browser', axis=1)
calls_noNa['Browser Categories'].value_counts()
```

Out[11]:

```
Chrome              214
Safari              176
Firefox or Opera     11
```

```
Name: Browser Categories, dtype: int64
```

In [12]:

```
#reduce state categories to PA or not PA
calls_noNa['State Categories'] = calls_noNa['State']
calls_noNa.loc[calls_noNa['State'] != 'PA', 'State Categories'] = 'Not PA'
calls_noNa = calls_noNa.drop('State', axis=1)
calls_noNa['State Categories'].value_counts()
```

Out[12]:

```
PA        312
Not PA     89
Name: State Categories, dtype: int64
```

In [13]:

```
#lump all google values
calls_noNa.loc[calls_noNa['Referrer'] == 'cse.google.com', 'Referrer'] =
'www.google.com'
```

In [14]:

```
categorical = ['Keywords Categories', 'Campaign', 'Page Categories', 'Device
Type', 'Browser Categories', 'Referrer', 'State Categories']
for i in categorical:
    dummies = pd.get_dummies(calls_noNa[i], drop_first=True)
    calls_noNa =
pd.concat([calls_noNa,dummies],axis=1).dropna().reset_index(drop=True)
    calls_noNa = calls_noNa.drop(i, axis=1)
calls_noNa = calls_noNa.astype(int)
```

In [15]:

```
#check for and fix multicollinearity

vif_df = pd.DataFrame()
vif_df["feature"] = calls_noNa.columns
vif_df["VIF"] = [variance_inflation_factor(calls_noNa.values, i)
                          for i in range(len(calls_noNa.columns))]

print(vif_df)
print('\n')

calls_noNa = calls_noNa.drop('Hour', axis=1)
calls_noNa = calls_noNa.drop('www.google.com', axis=1)

vif_df = pd.DataFrame()
vif_df["feature"] = calls_noNa.columns
vif_df["VIF"] = [variance_inflation_factor(calls_noNa.values, i)
                          for i in range(len(calls_noNa.columns))]

print(vif_df)
```

```
                                              feature        VIF
0                                            Duration   1.795709
```

1	sale	2.022827
2	Month	5.896868
3	Hour	13.241278
4	elder law attorney	1.213769
5	estate administration lawyer near me	1.554053
6	medicaid planning attorney	2.514512
7	pennsylvania lawyers	1.176284
8	pittsburgh probate lawyers	1.332965
9	probate and estate administration lawyers	1.351594
10	Estate Admin - Overall	1.555531
11	Estate Planning 005	3.392152
12	Medicaid - Elder Law - 001	3.481524
13	main/	2.136406
14	main/contact/	1.128137
15	main/firstname-redacted/	1.110253
16	main/pittsburgh-estate-planning-lawyer/power-o...	1.133315
17	mobile	4.683881
18	Firefox or Opera	1.175070
19	Safari	2.183395
20	www.google.com	11.865052
21	PA	4.678496

	feature	VIF
0	Duration	1.734367
1	sale	2.019141
2	Month	4.904542
3	elder law attorney	1.213154
4	estate administration lawyer near me	1.506437
5	medicaid planning attorney	2.483177
6	pennsylvania lawyers	1.171927
7	pittsburgh probate lawyers	1.291661
8	probate and estate administration lawyers	1.295143
9	Estate Admin - Overall	1.554796
10	Estate Planning 005	2.825811
11	Medicaid - Elder Law - 001	3.062890
12	main/	2.127636
13	main/contact/	1.119803
14	main/firstname-redacted/	1.106895
15	main/pittsburgh-estate-planning-lawyer/power-o...	1.124290
16	mobile	4.275562
17	Firefox or Opera	1.132177
18	Safari	2.144722
19	PA	4.038887

Perform Analysis

Feature Selection

In [16]:

```
def feature_selection(X,y,model,cv,score):

    features_names=pd.DataFrame(X.columns)

    #perform RFECV and fit model to get rankings
    rfecv = RFECV(model, step=1, cv=cv, scoring=score)
    fitted_values=rfecv.fit(X,y)
    ranks=pd.DataFrame(fitted_values.ranking_)

    rank=pd.concat([features_names,ranks], axis=1)
    rank.columns = ["Feature", "Rank"]

    #Select rank 1's
    most_important = rank.loc[rank['Rank'] ==1]
    most_important = most_important['Feature']

    print('Most important features ('+str(len(most_important))+')')
    print(str(most_important))

    return most_important
```

Analysis

In [17]:

```
def run_lr_model_split(df, n=5, score='f1', random_state=56):

    cv = StratifiedKFold(n_splits=n)

    #split dataset into dependent and independent variables
    features = list(df.columns)  #all columns but sale
    features.remove('sale')
    X = df[features].astype('category') # Features
    y = df['sale'] # independent var

    # split X and y into test and train
    X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,
random_state=random_state)

    # instantiate the model
```

```
    model = LogisticRegression(max_iter=10000)

    # fit the model
    model.fit(X_train,y_train)

    #predict
    y_pred=model.predict(X_test)

    #create confusion matrix
    cMat = metrics.confusion_matrix(y_test, y_pred)

    print(cMat)

    print('Accuracy score with train/test: %.3f' %
metrics.accuracy_score(y_test, model.predict(X_test))) #model accuracy
    print('Balanced Accuracy score with train/test: %.3f' %
metrics.balanced_accuracy_score(y_test, model.predict(X_test))) #balanced
model accuracy
    print('F1 score with train/test: %.3f' % metrics.f1_score(y_test,
model.predict(X_test))) #model accuracy
    print('ROC AUC with train/test: %.3f' % metrics.accuracy_score(y_test,
model.predict(X_test))) #model accuracy

    #metrics
    y_pred_prb = model.predict_proba(X_test)[::,1]
    fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_prb)

    #ROC curve plot
    plt.plot(fpr,tpr)
    plt.ylabel('True Positive Rate')
    plt.xlabel('False Positive Rate')
    plt.show()
```

In [18]:

```
def run_lr_model_cv(df, n=5, score='f1', random_state=56):

    cv = StratifiedKFold(n_splits=n)

    #split dataset into dependent and independent variables
    features = list(df.columns)  #all columns but sale
    features.remove('sale')
    X = df[features].astype('category') # Features
    y = df['sale'] # independent var

    # instantiate the model
    model = LogisticRegression(max_iter=10000)
```

```
    #Setting the range for class weights
    weights = np.linspace(0.0,0.99,100)
    #Creating a dictionary grid for grid search
    param_grid = {'class_weight': [{0:x, 1:1.0-x} for x in weights]}

    #Fitting grid search to the train data with cv to find class weights
    gridsearch = GridSearchCV(estimator= model,
                              param_grid= param_grid,
                              cv=cv,
                              n_jobs=-1,
                              scoring=score,
                              verbose=2).fit(X, y)

    weight_df = pd.DataFrame({ 'score':
gridsearch.cv_results_['mean_test_score'], 'weight': (1- weights)})
    optimum_weight = max(weight_df[weight_df['score'] ==
max(weight_df['score'])]['weight'])

    # re-instantiate the model
    model = LogisticRegression(max_iter=10000, random_state=random_state,
                                         class_weight={0: 1-optimum_weight, 1:
optimum_weight})

    #perform feature selection
    Best_X_Columns = feature_selection(X,y,model,n,score)
    X = X[Best_X_Columns]

    # fit the model
    #model.fit(X_train,y_train)

    #predict
    #y_pred=model.predict(X_test)

    # evaluate model with k-fold validation
    scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-
1)
    bal_scores = cross_val_score(model, X, y, scoring='balanced_accuracy',
cv=cv, n_jobs=-1)
    f1_scores = cross_val_score(model, X, y, scoring='f1', cv=cv, n_jobs=-1)
    w_f1_scores = cross_val_score(model, X, y, scoring='f1_weighted', cv=cv,
n_jobs=-1)
    auc_scores = cross_val_score(model, X, y, scoring='roc_auc', cv=cv,
n_jobs=-1)

    # k-fold validiation acuracy
```

```
    print('Accuracy with K-Fold validation: %.3f' % (mean(scores)))
    print('Balanced Accuracy with K-Fold validation: %.3f' %
(mean(bal_scores)))
    print('F1-Score with K-Fold validation: %.3f' % (mean(f1_scores)))
    print('ROC AUC with K-Fold validation: %.3f' % (mean(auc_scores)))

    return model
```

In [19]:

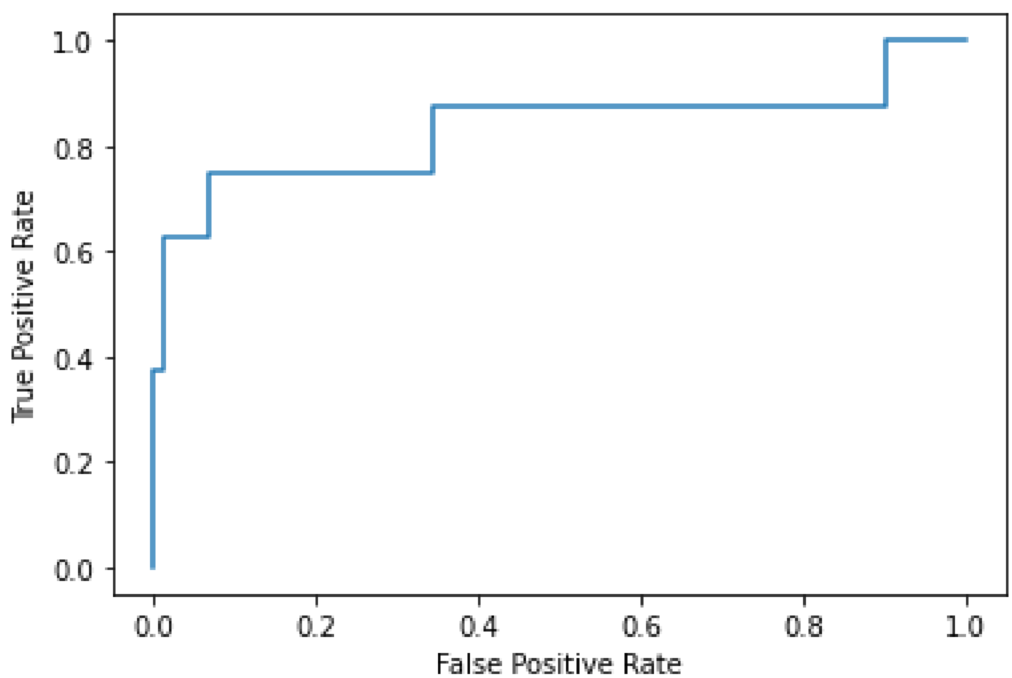
```
seed = np.random.randint(1000)
seed
```

Out[19]:

```
204
```

In [20]:

```
#linear regression, F1, CV = 5
run_lr_model_split(calls_noNa)
[[72  1]
 [ 5  3]]
Accuracy score with train/test: 0.926
Balanced Accuracy score with train/test: 0.681
F1 score with train/test: 0.500
ROC AUC with train/test: 0.926
```



In [21]:

```
#linear regression, F1, CV = 5
run_lr_model_cv(calls_noNa)
Fitting 5 folds for each of 100 candidates, totalling 500 fits
Most important features (19)
0                                                  Duration
```

```
1                                                   Month
2                                      elder law attorney
3                        estate administration lawyer near me
4                                   medicaid planning attorney
5                                         pennsylvania lawyers
6                                   pittsburgh probate lawyers
7                   probate and estate administration lawyers
8                                        Estate Admin - Overall
9                                          Estate Planning 005
10                                  Medicaid - Elder Law - 001
11                                                      main/
12                                              main/contact/
13                                   main/firstname-redacted/
14    main/pittsburgh-estate-planning-lawyer/power-o...
15                                                     mobile
16                                          Firefox or Opera
17                                                     Safari
18                                                         PA
Name: Feature, dtype: object
Accuracy with K-Fold validation: 0.918
Balanced Accuracy with K-Fold validation: 0.719
F1-Score with K-Fold validation: 0.576
ROC AUC with K-Fold validation: 0.877
```

Out[21]:

```
LogisticRegression(class_weight={0: 0.4, 1: 0.6}, max_iter=10000,
                   random_state=56)
```

In [22]:

```
#linear regression, F1, CV = 10
run_lr_model_cv(calls_noNa, n=10)
```

```
Fitting 10 folds for each of 100 candidates, totalling 1000 fits
Most important features (19)
0                                                Duration
1                                                   Month
2                                      elder law attorney
3                        estate administration lawyer near me
4                                   medicaid planning attorney
5                                         pennsylvania lawyers
6                                   pittsburgh probate lawyers
7                   probate and estate administration lawyers
8                                        Estate Admin - Overall
9                                          Estate Planning 005
10                                  Medicaid - Elder Law - 001
11                                                      main/
12                                              main/contact/
13                                   main/firstname-redacted/
14    main/pittsburgh-estate-planning-lawyer/power-o...
```

```
15                                                   mobile
16                                         Firefox or Opera
17                                                   Safari
18                                                       PA
Name: Feature, dtype: object
Accuracy with K-Fold validation: 0.898
Balanced Accuracy with K-Fold validation: 0.788
F1-Score with K-Fold validation: 0.562
ROC AUC with K-Fold validation: 0.897
```

Out[22]:

```
LogisticRegression(class_weight={0: 0.19999999999999996, 1: 0.8},
                   max_iter=10000, random_state=56)
```

In [23]:

```
def run_rf_model(df, n=5, score='f1', random_state=56):

    cv = StratifiedKFold(n_splits=n)

    #split dataset into dependent and independent variables
    features = list(df.columns)  #all columns but sale
    features.remove('sale')
    X = df[features].astype('category') # Features
    y = df['sale'] # independent var

    # instantiate the model
    model = RandomForestClassifier(random_state=random_state)

    # Number of trees in random forest
    n_estimators = [int(x) for x in np.linspace(50, 2000, num = 20)] # The
number of trees in the forest.
    max_features = ['auto', 'sqrt'] # The number of features to consider when
looking for the best split
    max_depth = [int(x) for x in np.linspace(20, 120, num = 10)] # The
maximum depth of the tree.
    min_samples_split = [2, 4, 6, 8, 10] # The minimum number of samples
required to split an internal node
    min_samples_leaf = [1, 2, 3, 4, 5] # The minimum number of samples
required to be at a leaf node.
    bootstrap = [True, False] # Whether bootstrap samples are used when
building trees.

    random_grid = {'n_estimators': n_estimators,
    'max_features': max_features,
    'max_depth': max_depth,
    'min_samples_split': min_samples_split,
    'min_samples_leaf': min_samples_leaf,
    'bootstrap': bootstrap}
```

```
    # Create the random grid
    param_grid = {'n_estimators': n_estimators,
                  'max_depth': max_depth,
                  'min_samples_split': min_samples_split,
                  'min_samples_leaf': min_samples_leaf}
    model = RandomizedSearchCV(estimator = model, param_distributions =
param_grid, cv = 5,
                                   verbose=2, n_jobs = -1, n_iter = 250,
scoring='f1')

    # fit the model
    model.fit(X,y)
    print(model.best_params_)

    # evaluate model with k-fold validation
    scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-
1)
    bal_scores = cross_val_score(model, X, y, scoring='balanced_accuracy',
cv=cv, n_jobs=-1)
    f1_scores = cross_val_score(model, X, y, scoring='f1', cv=cv, n_jobs=-1)
    w_f1_scores = cross_val_score(model, X, y, scoring='f1_weighted', cv=cv,
n_jobs=-1)
    auc_scores = cross_val_score(model, X, y, scoring='roc_auc', cv=cv,
n_jobs=-1)

    # k-fold validiation acuracy
    print('Accuracy with K-Fold validation: %.3f' % (mean(scores)))
    print('Balanced Accuracy with K-Fold validation: %.3f' %
(mean(bal_scores)))
    print('F1-Score with K-Fold validation: %.3f' % (mean(f1_scores)))
    print('ROC AUC with K-Fold validation: %.3f' % (mean(auc_scores)))
```

In [69]:

```
#random forest model, f1, CV = 5
run_rf_model(calls_noNa, n=5, score='f1', random_state=56)
Fitting 5 folds for each of 250 candidates, totalling 1250 fits
{'n_estimators': 50, 'min_samples_split': 10, 'min_samples_leaf': 3,
'max_depth': 97}
Accuracy with K-Fold validation: 0.895
Balanced Accuracy with K-Fold validation: 0.728
F1-Score with K-Fold validation: 0.510
ROC AUC with K-Fold validation: 0.843
```

In [ ]:

In [ ]:

Part 4: Recommender System

Code for recommender system using KNN to find similar zip codes

bazalewski_capstone_recommender

April 28, 2022

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

#from scipy.spatial import distance
#from sklearn.metrics.pairwise import linear_kernel

from sklearn.neighbors import NearestNeighbors
import numpy as np

from statsmodels.stats.outliers_influence import variance_inflation_factor
```

[2]:
```
census_df = pd.read_csv('cleaned_census.csv')
census_df = census_df.drop('Unnamed: 0', axis=1)
census_df = census_df.replace('-',np.NAN)
census_df = census_df.replace('+','')
census_df = census_df.dropna().reset_index(drop=True)
```

[3]:
```
census_df.columns
```

[3]:
```
Index(['ZCTA', 'Total Households', 'Percent Married Couple Family',
       'Percent Married Couple Family with Children',
       'Percent Male Householder', 'Percent Female Householder',
       'Average Household Size', 'Average Family Size',
       'Percent Males Never Married', 'Percent Males Married',
       'Percent Males Divorced', 'Percent Females Never Married',
       'Percent Females Married', 'Percent Females Divorced',
       'Percent High School Grad', 'Percent Assoc Deg',
       'Percent Bachelors Deg', 'Percent Graduate Deg', 'Percent Disabled',
       'Total Pop 16 and Up', 'Percent in Labor Force', 'Unemployment Rate',
       'Percent Private Sector', 'Percent Govt Workers',
       'Percent Self Employed', 'Median Income', 'Mean Income',
       'Per Capita Income', 'Percent 2 Bedroom Homes',
       'Percent 3 Bedroom Homes', 'Percent 4 Bedroom Homes',
       'Median House Value', 'Median Mortgage',
       'Tot Housing Units with Mortgage',
```

```
       'Mortgage Less than 20 Percent of Income',
       'Mortgage Between 20 and 25 Percent of Income',
       'Mortgage Between 25 and 30 Percent of Income',
       'Mortgage Between 30 and 35 Percent of Income',
       'Mortgage More than 35 Percent of Income', 'Total Units Paying Rent',
       'Rent Less than 15 Percent of Income',
       'Rent Between 15 and 20 Percent of Income',
       'Rent Between 20 and 25 Percent of Income',
       'Rent Between 20 and 25 Percent of Income.1',
       'Rent Between 25 and 30 Percent of Income',
       'Rent Between 30 and 35 Percent of Income',
       'Rent More than 35 Percent of Income', 'Total Pop', 'Percent Male',
       'Percent Female', 'Median Age', 'Percent Under 18',
       'Percent 62 and Over', 'Percent 65 and Over', 'Percent White',
       'Percent Black', 'Percent Asian', 'Percent Hispanic'],
      dtype='object')
```

[4]:
```
state_zip_df = pd.read_csv('state_zip.csv')
census_df = census_df.
 ↪merge(state_zip_df,how='left',left_on='ZCTA',right_on='Zipcode')
```

[5]:
```
census_df_labels = census_df[['ZCTA','City','State']]

census_df_labels
```

[5]:
```
        ZCTA         City State
0       1001       AGAWAM    MA
1       1002      AMHERST    MA
2       1005        BARRE    MA
3       1007  BELCHERTOWN    MA
4       1010    BRIMFIELD    MA
…        …          …   …
26111  99919   THORNE BAY    AK
26112  99921        CRAIG    AK
26113  99925      KLAWOCK    AK
26114  99926    METLAKATLA    AK
26115  99929     WRANGELL    AK

[26116 rows x 3 columns]
```

[6]:
```
census_df = census_df.loc[:,(census_df.columns!='ZCTA')&
                                  (census_df.columns!='City') &
                                  (census_df.columns!='State')]
census_df = census_df.astype(float)
```

[7]:
```
#check for and fix multicollinearity
```

```python
vif_df = pd.DataFrame()

vif_df["feature"] = census_df.columns

vif_df["VIF"] = [variance_inflation_factor(census_df.values, i)
                          for i in range(len(census_df.columns))]

print(vif_df)
print('\n')
```

```
                                      feature           VIF
0                            Total Households  2.492275e+02
1               Percent Married Couple Family  7.595637e+02
2   Percent Married Couple Family with Children  3.981199e+01
3                     Percent Male Householder  6.289623e+01
4                   Percent Female Householder  1.105114e+02
5                      Average Household Size  7.348391e+02
6                         Average Family Size  5.989733e+02
7                 Percent Males Never Married  1.990573e+02
8                       Percent Males Married  7.146141e+02
9                      Percent Males Divorced  2.846997e+01
10              Percent Females Never Married  7.852323e+01
11                    Percent Females Married  4.838344e+02
12                   Percent Females Divorced  2.005092e+01
13                   Percent High School Grad  4.855859e+01
14                         Percent Assoc Deg  1.086419e+01
15                     Percent Bachelors Deg  2.434007e+01
16                      Percent Graduate Deg  1.380715e+01
17                          Percent Disabled  2.026857e+01
18                       Total Pop 16 and Up  7.836541e+02
19                     Percent in Labor Force  1.512342e+02
20                         Unemployment Rate  4.611772e+00
21                    Percent Private Sector  1.106870e+04
22                      Percent Govt Workers  5.283610e+02
23                     Percent Self Employed  1.546322e+02
24                             Median Income  1.250984e+02
25                               Mean Income  3.095268e+02
26                         Per Capita Income  2.187790e+02
27                    Percent 2 Bedroom Homes  3.004610e+01
28                    Percent 3 Bedroom Homes  5.022813e+01
29                    Percent 4 Bedroom Homes  1.882163e+01
30                         Median House Value  1.490568e+01
31                           Median Mortgage  7.137184e+01
32             Tot Housing Units with Mortgage  5.900618e+01
33        Mortgage Less than 20 Percent of Income  6.091782e+05
34   Mortgage Between 20 and 25 Percent of Income  6.688573e+04
35   Mortgage Between 25 and 30 Percent of Income  3.244927e+04
```

```
36  Mortgage Between 30 and 35 Percent of Income  1.665414e+04
37       Mortgage More than 35 Percent of Income  1.301769e+05
38                       Total Units Paying Rent  3.226654e+01
39           Rent Less than 15 Percent of Income  3.758597e+01
40      Rent Between 15 and 20 Percent of Income  1.163395e+05
41      Rent Between 20 and 25 Percent of Income  6.835094e+04
42    Rent Between 20 and 25 Percent of Income.1  5.620552e+04
43      Rent Between 25 and 30 Percent of Income  4.251237e+04
44      Rent Between 30 and 35 Percent of Income  2.760454e+04
45           Rent More than 35 Percent of Income  3.182756e+05
46                                     Total Pop  5.592547e+02
47                                  Percent Male  1.155990e+06
48                                Percent Female  1.171290e+06
49                                    Median Age  2.795888e+02
50                              Percent Under 18  9.047179e+01
51                           Percent 62 and Over  2.489791e+02
52                           Percent 65 and Over  1.771691e+02
53                                 Percent White  1.299885e+02
54                                 Percent Black  6.580474e+00
55                                 Percent Asian  3.148621e+00
56                              Percent Hispanic  3.426226e+00
57                                       Zipcode  7.209993e+00
```

[8]:
```
census_df_subset = census_df[[
       'Percent Married Couple Family with Children',
       'Percent Males Divorced',
       'Percent Bachelors Deg',
       'Percent Disabled',
       'Unemployment Rate',
       'Percent Govt Workers',
       'Percent Self Employed',
       'Percent 4 Bedroom Homes',
       'Median House Value',
       'Total Pop',
       'Percent Black', 'Percent Asian', 'Percent Hispanic',
       'Mortgage Between 20 and 25 Percent of Income',
       'Mortgage Between 30 and 35 Percent of Income',
       'Mortgage More than 35 Percent of Income',
       'Rent Between 15 and 20 Percent of Income',
       'Rent Between 20 and 25 Percent of Income',
       'Rent Between 25 and 30 Percent of Income',
       'Rent Between 30 and 35 Percent of Income']]

vif_df = pd.DataFrame()
vif_df["feature"] = census_df_subset.columns
```

```
vif_df["VIF"] = [variance_inflation_factor(census_df_subset.values, i)
                          for i in range(len(census_df_subset.columns))]

print(vif_df)
```

```
                                        feature        VIF
0    Percent Married Couple Family with Children  10.183023
1                         Percent Males Divorced   7.270229
2                          Percent Bachelors Deg   9.199497
3                               Percent Disabled  10.461332
4                              Unemployment Rate   3.878860
5                           Percent Govt Workers   5.254223
6                          Percent Self Employed   3.798134
7                        Percent 4 Bedroom Homes   7.984256
8                             Median House Value   5.145396
9                                      Total Pop   2.808340
10                                 Percent Black   1.704563
11                                 Percent Asian   1.878822
12                              Percent Hispanic   1.998047
13  Mortgage Between 20 and 25 Percent of Income   5.520180
14  Mortgage Between 30 and 35 Percent of Income   2.813584
15       Mortgage More than 35 Percent of Income   6.729897
16      Rent Between 15 and 20 Percent of Income   3.164907
17      Rent Between 20 and 25 Percent of Income   2.859721
18      Rent Between 25 and 30 Percent of Income   2.691519
19      Rent Between 30 and 35 Percent of Income   2.278236
```

[9]:

```
def find_KNN(df,df_y,knn,lookup,state='All'):

    if state != 'All':
        df = pd.concat([df.loc[df_y['State']==state],df.
 ↪loc[df_y['ZCTA']==lookup]])
        df_y = pd.
 ↪concat([df_y[df_y['State']==state],df_y[df_y['ZCTA']==lookup]]).
 ↪reset_index(drop=True)
        df = df.reset_index(drop=True)

    X = df.to_numpy()
    nbrs = NearestNeighbors(n_neighbors=knn, algorithm='ball_tree').fit(X)
    distances, indices = nbrs.kneighbors(X)

    df_y[df_y['ZCTA']==lookup]
    i = df_y[df_y['ZCTA']==lookup].index.values[0]
    zips = df_y.iloc[indices[i][1:knn+1]]
    print(zips)
    return zips
```

0.1 Tests

[10]:
```
#full model test, Brooklyn zip code
zips = find_KNN(census_df,census_df_labels,5,11201)
```

```
       ZCTA           City State
2091  10019       NEW YORK    NY
5372  22102        MC LEAN    VA
368    2445      BROOKLINE    MA
381    2467  CHESTNUT HILL    MA
```

[11]:
```
#full model test, Brooklyn zip code, Virgina results
zips = find_KNN(census_df,census_df_labels,5,11201,'VA')
```

```
     ZCTA       City State
58  22102    MC LEAN    VA
79  22207  ARLINGTON    VA
69  22182     VIENNA    VA
77  22205  ARLINGTON    VA
```

[12]:
```
#variable subset test, Brooklyn zip code
zips = find_KNN(census_df_subset,census_df_labels,5,11201)
```

```
        ZCTA         City State
23638  90019  LOS ANGELES    CA
23940  92024    ENCINITAS    CA
25149  96816     HONOLULU    HI
2306   11221     BROOKLYN    NY
```

[13]:
```
#variable subset test, Brooklyn zip code, Virgina results
zips = find_KNN(census_df_subset,census_df_labels,5,11201,'VA')
```

```
      ZCTA       City State
79   22207  ARLINGTON    VA
58   22102    MC LEAN    VA
57   22101    MC LEAN    VA
439  24011    ROANOKE    VA
```

0.2 Suggested Areas

[14]:
```
zip_codes = [15317,15227]

states = ['NY','NJ','OH','WV','MD','VA','NC','SC','GA','FL']

zips = pd.DataFrame()
```

[15]:
```
#variable subset, All States
for i in zip_codes:
```

```
    find_KNN(census_df_subset,census_df_labels,5,i)
```

```
        ZCTA           City State
24728  95301        ATWATER    CA
22100  80014         AURORA    CO
8318   33060  POMPANO BEACH    FL
24045  92307   APPLE VALLEY    CA
        ZCTA          City State
2521   12010     AMSTERDAM    NY
19390  70607  LAKE CHARLES    LA
10904  43512      DEFIANCE    OH
17084  62226    BELLEVILLE    IL
```

[16]:

```
#variable subset, selected states
for i in states:
    for j in zip_codes:
        zips = pd.
 ↪concat([zips,find_KNN(census_df_subset,census_df_labels,5,j,i)])
```

```
       ZCTA          City State
1252  14534     PITTSFORD    NY
655   12603  POUGHKEEPSIE    NY
632   12553   NEW WINDSOR    NY
437   11967       SHIRLEY    NY
       ZCTA          City State
446   12010     AMSTERDAM    NY
803   13045      CORTLAND    NY
1293  14623     ROCHESTER    NY
555   12304   SCHENECTADY    NY
     ZCTA          City State
337  8080         SEWELL    NJ
317  8054   MOUNT LAUREL    NJ
91   7201      ELIZABETH    NJ
249  7860         NEWTON    NJ
     ZCTA                 City State
472  8759  MANCHESTER TOWNSHIP    NJ
301  8030      GLOUCESTER CITY    NJ
376  8232         PLEASANTVILLE    NJ
362  8110            PENNSAUKEN    NJ
      ZCTA        City State
426  44145    WESTLAKE    OH
92   43201    COLUMBUS    OH
99   43209    COLUMBUS    OH
763  45244  CINCINNATI    OH
      ZCTA        City State
188  43512    DEFIANCE    OH
759  45240  CINCINNATI    OH
521  44460       SALEM    OH
```

225	43606	TOLEDO	OH

	ZCTA	City	State
266	26508	MORGANTOWN	WV
95	25414	CHARLES TOWN	WV
90	25403	MARTINSBURG	WV
102	25430	KEARNEYSVILLE	WV

	ZCTA	City	State
146	25701	HUNTINGTON	WV
150	25705	HUNTINGTON	WV
151	25801	BECKLEY	WV
4	24740	PRINCETON	WV

	ZCTA	City	State
78	20785	HYATTSVILLE	MD
61	20748	TEMPLE HILLS	MD
141	21060	GLEN BURNIE	MD
77	20784	HYATTSVILLE	MD

	ZCTA	City	State
192	21217	BALTIMORE	MD
216	21502	CUMBERLAND	MD
221	21532	FROSTBURG	MD
204	21229	BALTIMORE	MD

	ZCTA	City	State
303	23321	CHESAPEAKE	VA
168	22801	HARRISONBURG	VA
305	23323	CHESAPEAKE	VA
131	22602	WINCHESTER	VA

	ZCTA	City	State
586	24501	LYNCHBURG	VA
356	23607	NEWPORT NEWS	VA
440	24012	ROANOKE	VA
394	23847	EMPORIA	VA

	ZCTA	City	State
577	28607	BOONE	NC
350	28105	MATTHEWS	NC
478	28411	WILMINGTON	NC
124	27511	CARY	NC

	ZCTA	City	State
609	28658	NEWTON	NC
373	28150	SHELBY	NC
340	28086	KINGS MOUNTAIN	NC
374	28152	SHELBY	NC

	ZCTA	City	State
135	29414	CHARLESTON	SC
290	29715	FORT MILL	SC
86	29205	COLUMBIA	SC
7	29016	BLYTHEWOOD	SC

	ZCTA	City	State
304	29801	AIKEN	SC

```
244  29640      EASLEY    SC
146  29440  GEORGETOWN    SC
70   29154      SUMTER    SC
      ZCTA     City State
174  30316  ATLANTA    GA
356  30809    EVANS    GA
11   30019   DACULA    GA
50   30082   SMYRNA    GA
      ZCTA            City State
6    30012         CONYERS    GA
54   30088  STONE MOUNTAIN    GA
391  31021          DUBLIN    GA
472  31404        SAVANNAH    GA
      ZCTA           City State
377  33060  POMPANO BEACH    FL
440  33183          MIAMI    FL
347  33014        HIALEAH    FL
116  32309    TALLAHASSEE    FL
      ZCTA              City State
605  33709  SAINT PETERSBURG    FL
315  32905          PALM BAY    FL
5    32025         LAKE CITY    FL
798  34472             OCALA    FL
```

[17]:
```
zips
```

[17]:
```
       ZCTA              City State
1252  14534         PITTSFORD    NY
655   12603      POUGHKEEPSIE    NY
632   12553       NEW WINDSOR    NY
437   11967           SHIRLEY    NY
446   12010         AMSTERDAM    NY
…     …                     …   …
116   32309       TALLAHASSEE    FL
605   33709  SAINT PETERSBURG    FL
315   32905          PALM BAY    FL
5     32025         LAKE CITY    FL
798   34472             OCALA    FL

[80 rows x 3 columns]
```

[18]:
```
zips.to_csv('recommended_zips.csv')
```

[ ]:
```
```