In [ ]:
```
print("prelim stats")
```

```
prelim stats
```

In [ ]:
```
#import libraries

import csv
import pandas as pd
import numpy as np
```

In [ ]:
```
#import and read csv file
#display first 10 rows

import pandas as pd
crime_data1 = pd.read_csv(r"C:\Users\radon\Documents\CIND820/crimedata.csv")

crime_data1.head(10)
```

Out[ ]:

	communityname	state	countyCode	communityCode	fold	population	householdsize	race
0	BerkeleyHeightstownship	NJ	39	5320	1	11980	3.10	
1	Marpletownship	PA	45	47616	1	23123	2.82	
2	Tigardcity	OR	?	?	1	29344	2.43	
3	Gloversvillecity	NY	35	29443	1	16656	2.40	
4	Bemidjicity	MN	7	5068	1	11245	2.76	
5	Springfieldcity	MO	?	?	1	140494	2.45	
6	Norwoodtown	MA	21	50250	1	28700	2.60	
7	Andersoncity	IN	?	?	1	59459	2.45	
8	Fargocity	ND	17	25700	1	74111	2.46	
9	Wacocity	TX	?	?	1	103590	2.62	

In [ ]:
```
#display data types of all columns

datatypes = crime_data1.dtypes
print(datatypes)
```

In [ ]:
```
#replace all missing values with NaN

import pandas as pd
crime_data = pd.read_csv('crimedata.csv', na_values=['?'])
crime_data.head(10)
```

Out[ ]:

	communityname	state	countyCode	communityCode	fold	population	householdsize	race
0	BerkeleyHeightstownship	NJ	39.0	5320.0	1	11980	3.10	
1	Marpletownship	PA	45.0	47616.0	1	23123	2.82	
2	Tigardcity	OR	NaN	NaN	1	29344	2.43	
3	Gloversvillecity	NY	35.0	29443.0	1	16656	2.40	
4	Bemidjicity	MN	7.0	5068.0	1	11245	2.76	
5	Springfieldcity	MO	NaN	NaN	1	140494	2.45	
6	Norwoodtown	MA	21.0	50250.0	1	28700	2.60	
7	Andersoncity	IN	NaN	NaN	1	59459	2.45	
8	Fargocity	ND	17.0	25700.0	1	74111	2.46	
9	Wacocity	TX	NaN	NaN	1	103590	2.62	

In [ ]:
```
#check number of rows
len(crime_data)
```

Out[ ]:
```
2215
```

In [ ]:
```
#check datatypes again

datatypes2 = crime_data.dtypes
print(datatypes2)
```

In [ ]:
```
#display the sum of all NaN in each column

crime_data.isnull().sum()
```

In [ ]:
```
#seperate all columns with missing values

null_cols = crime_data[crime_data.columns[crime_data.isna().any()]]
null_cols.head()
```

Out[ ]:

	countyCode	communityCode	OtherPerCap	LemasSwornFT	LemasSwFTPerPop	LemasSwFTFieldOps
0	39.0	5320.0	5115.0	NaN	NaN	NaN
1	45.0	47616.0	5250.0	NaN	NaN	NaN
2	NaN	NaN	5954.0	NaN	NaN	NaN
3	35.0	29443.0	2451.0	NaN	NaN	NaN
4	7.0	5068.0	3000.0	NaN	NaN	NaN

In [ ]:
```
#display sums of missing values of only columns that contain missing values

null_list= null_cols.columns.values
null_cols.isnull().sum()
```

In [ ]:
```
#change settings to display all columns
pd.set_option('display.max_columns', None)

#get summary statistics of all columns
crime_data.describe()
```

Out[ ]:

	countyCode	communityCode	fold	population	householdsize	racepctblack	racePct'
count	994.000000	991.000000	2215.000000	2.215000e+03	2215.000000	2215.000000	2215.0
mean	65.587525	45209.251261	5.494357	5.311798e+04	2.707327	9.335102	83.9
std	117.831399	25425.861573	2.872924	2.046203e+05	0.334120	14.247156	16.4
min	1.000000	70.000000	1.000000	1.000500e+04	1.600000	0.000000	2.6
25%	11.000000	22887.000000	3.000000	1.436600e+04	2.500000	0.860000	76.3
50%	27.000000	46925.000000	5.000000	2.279200e+04	2.660000	2.870000	90.3
75%	80.500000	65805.000000	8.000000	4.302400e+04	2.850000	11.145000	96.2
max	840.000000	94597.000000	10.000000	7.322564e+06	5.280000	96.670000	99.6

In [ ]:
```
crime_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2215 entries, 0 to 2214
Columns: 147 entries, communityname to nonViolPerPop
dtypes: float64(116), int64(29), object(2)
memory usage: 2.5+ MB
```

In [ ]:
```
#find the mean of each column by state
mean_data = crime_data.groupby("state").mean()
mean_data.head()
```

```
C:\Users\radon\AppData\Local\Temp\ipykernel_10788\3113613801.py:2: FutureWarning: The
default value of numeric_only in DataFrameGroupBy.mean is deprecated. In a future ver
sion, numeric_only will default to False. Either specify numeric_only or select only
columns which should be valid for the function.
  mean_data = crime_data.groupby("state").mean()
```

Out[ ]:

state	countyCode	communityCode	fold	population	householdsize	racepctblack	racePctWhi
AK	NaN	NaN	7.333333	94644.000000	2.756667	6.826667	77.9066
AL	NaN	NaN	6.000000	39231.186047	2.611860	27.044651	71.7995
AR	NaN	NaN	5.440000	32674.520000	2.652800	19.470000	78.8836
AZ	NaN	NaN	5.700000	125811.650000	2.778500	2.610500	84.6555
CA	76.0	93325.0	5.591398	79654.179211	2.926129	5.312222	71.9737

In [ ]:
```
#find the mean of crime per pop columns by state
state_crimes =crime_data.groupby("state")["murdPerPop", "rapesPerPop", "robbbPerPop",
state_crimes.head(10)
```

```
C:\Users\radon\AppData\Local\Temp\ipykernel_10788\76998245.py:2: FutureWarning: Index
ing with multiple keys (implicitly converted to a tuple of keys) will be deprecated,
use a list instead.
  state_crimes =crime_data.groupby("state")["murdPerPop", "rapesPerPop", "robbbPerPo
p", "assaultPerPop", "burglPerPop", "larcPerPop", "autoTheftPerPop", "arsonsPerPop"].
mean()
```

Out[ ]:

state	murdPerPop	rapesPerPop	robbbPerPop	assaultPerPop	burglPerPop	larcPerPop	autoTheftPe
AK	8.870000	66.080000	156.103333	345.516667	639.463333	3813.090000	485.28
AL	11.342093	42.825349	199.333721	777.197674	1273.725814	3730.038605	399.10
AR	10.735600	52.083600	195.645600	460.085200	1450.826400	4575.336800	450.89
AZ	4.613500	30.195000	119.256000	397.030000	1510.476500	4325.365000	661.16
CA	8.134588	32.646882	273.137276	496.704964	1248.996595	3022.680896	819.77
CO	3.822800	41.320400	84.554000	397.163600	965.632800	4010.375600	350.70
CT	3.841972	16.033239	111.564366	127.755217	740.837746	2372.257465	425.04
DC	81.950000	58.480000	1282.850000	1625.090000	2081.590000	5679.780000	1454.87
DE	0.000000	123.330000	267.210000	496.750000	859.880000	6118.530000	260.36
FL	7.305556	49.775889	359.934778	742.029556	1932.676667	5190.722333	866.24

In [ ]:
```
#find the states with min and max values for crimes
state_crimes["murdPerPop"].sort_values()
state_crimes["rapesPerPop"].sort_values()
state_crimes["robbbPerPop"].sort_values()
state_crimes["assaultPerPop"].sort_values()
state_crimes["burglPerPop"].sort_values()
state_crimes["larcPerPop"].sort_values()
state_crimes["autoTheftPerPop"].sort_values()
state_crimes["arsonsPerPop"].sort_values()
```

In [ ]:
```
#find representation of each state in dataset, how many rows does each state have?
crime_data.groupby(['state'])['state'].count().sort_values()
```

In [ ]:
```
#Examine the STATE with the overall max values of each crime
crime_data.loc[crime_data['state'] == 'DC']
```

Out[ ]:

	communityname	state	countyCode	communityCode	fold	population	householdsize	racepct
1581	Washingtoncity	DC	1.0	50000.0	8	606900	2.43	

In [ ]:
```
#Sort the dataset by population
crime_data.sort_values(['population'])
```

In [ ]:
```
#Subset the states with the most representation in the dataset
max_rep = ['CA', 'NJ', 'TX']

maxrep_subset = crime_data[crime_data.state.isin(max_rep)]

maxrep_subset.describe()
```

Out[ ]:

	countyCode	communityCode	fold	population	householdsize	racepctblack	racePctW
count	213.000000	213.000000	652.000000	6.520000e+02	652.000000	652.000000	652.00
mean	19.215962	45170.629108	5.624233	5.946048e+04	2.855675	7.809647	77.52
std	13.235541	23379.476968	2.891520	1.761253e+05	0.408726	11.403043	16.83
min	1.000000	70.000000	1.000000	1.002300e+04	1.600000	0.000000	7.21
25%	7.000000	25770.000000	3.000000	1.559650e+04	2.610000	1.275000	68.76
50%	21.000000	45990.000000	6.000000	2.676400e+04	2.795000	3.425000	81.35
75%	27.000000	63360.000000	8.000000	5.233275e+04	3.040000	8.917500	90.52
max	81.000000	93480.000000	10.000000	3.485398e+06	5.280000	89.950000	99.17

In [ ]:
```
#Subset the states with the least representation
min_rep = ['KS', 'DE', 'DC', 'AK', 'VT']

minrep_subset = crime_data[crime_data.state.isin(min_rep)]

minrep_subset.describe()
```

Out[ ]:

	countyCode	communityCode	fold	population	householdsize	racepctblack	racePctW
count	6.000000	6.000000	10.000000	10.000000	10.000000	10.000000	10.00
mean	36.000000	46991.666667	5.600000	130737.000000	2.644000	13.102000	80.21
std	67.441827	28123.129567	2.458545	195996.657595	0.183073	20.853535	21.28
min	1.000000	10675.000000	2.000000	12809.000000	2.420000	0.390000	29.60
25%	7.000000	23656.250000	4.000000	20360.250000	2.470000	0.670000	74.42
50%	7.000000	55612.500000	5.500000	29236.500000	2.685000	3.760000	81.51
75%	17.500000	64937.500000	7.000000	179535.250000	2.777500	12.540000	96.97
max	173.000000	79000.000000	10.000000	606900.000000	2.920000	65.840000	98.93

* MODULE 3: INITIAL CODE AND RESULTS

In [ ]:
```
#az = crime_data.loc[crime_data['state'] == "CA"]
#az.boxplot(column="murdPerPop")

#dc = crime_data.loc[crime_data['state']== "DC"]
```

Previously, we have discovered a number of missing values in the dataset. The number of values missing in each column varies. At this point, there are missing number values which are significant and cannot be used in the analysis. These columns will have to be dropped. I have decided on a NaN threshold of 50% (0.5), so any columns which are missing 50% of its values or more will be dropped from the dataset.

In [ ]:
```
#after deciding on a NaN ratio threshold, drop the columns which exceed the threshold

crimedata_reduced= crime_data.drop(['countyCode', 'communityCode', 'fold', 'LemasSworr
crimedata_reduced.head()
```

Out[ ]:

	communityname	state	population	householdsize	racepctblack	racePctWhite	racePctAsian
0	BerkeleyHeightstownship	NJ	11980	3.10	1.37	91.78	6.50
1	Marpletownship	PA	23123	2.82	0.80	95.57	3.44
2	Tigardcity	OR	29344	2.43	0.74	94.33	3.43
3	Gloversvillecity	NY	16656	2.40	1.70	97.35	0.50
4	Bemidjicity	MN	11245	2.76	0.53	89.16	1.17

First, I will decide on my target/dependent variables. The two I have chosen are MURDERS and ROBBERIES. Many of my potential dependent variables are missing values, and are not able to be imputed as it may introduce bias. For this reason, I have chosen these 2 categories as my dependent variables because they have the LEAST amount of missing values. Now that my dependent variables are chosen, I can move on to dealing with my independent variables.

There are still a number of columns that remain with missing values. The number of NaNs, however, were below the threshold of 0.5 so they will be used in the analysis. To deal with these missing values, we will turn to imputation. For the columns of our independent variables with 1-15 total missing values, I have chosen to impute them with the overall mode of the column. Because the distribution of the columns is skewed, I have chosen to impute with the mode value per each individual column.

However, there are still a number of columns with a significant number of missing values, but fall below the threshold. These will have to be dealt with differently. For these values, I have chosen to aggregate the data first, and then choose my imputation method. The nature of the organization of the crimes is by neighbourhood, which are each found in a state. I thought aggregrating by state was the most logical choice, as we may find more geographical, social, and economic similarties overall (but not entirely) within each state and its population. Because there are a different number of neighbourhoods represented for each state, the representation is unbalanced. Because of this, I thought the most appropriate imputation method would be to impute these missing values with the mean by state, per each column.

In [ ]:
```
#impute missing values
#for columns with very small number of missing values, impute with MODE

crimedata_reduced['burglaries'].fillna(crimedata_reduced['burglaries'].mode()[0], inpl
crimedata_reduced['burglPerPop'].fillna(crimedata_reduced['burglPerPop'].mode()[0], ir
crimedata_reduced['larcenies'].fillna(crimedata_reduced['larcenies'].mode()[0], inplac
crimedata_reduced['larcPerPop'].fillna(crimedata_reduced['larcPerPop'].mode()[0], inpl
crimedata_reduced['autoTheft'].fillna(crimedata_reduced['autoTheft'].mode()[0], inplac
crimedata_reduced['autoTheftPerPop'].fillna(crimedata_reduced['autoTheftPerPop'].mode(
crimedata_reduced['OtherPerCap'].fillna(crimedata_reduced['OtherPerCap'].mode()[0], ir
crimedata_reduced['assaults'].fillna(crimedata_reduced['assaults'].mode()[0], inplace=
crimedata_reduced['assaultPerPop'].fillna(crimedata_reduced['assaultPerPop'].mode()[0]
```

In [ ]:
```
#Find out the total counts of how many times each state is represented in the dataset
#With this information, I have chosen to impute by the mean value of the column, aggre

#crimedata_reduced[crimedata_reduced['rapes'].isnull()]
#crimedata_reduced.pivot_table(index = ['state'], aggfunc ='size')

from collections import Counter
print(Counter(crimedata_reduced['state']))
```

```
Counter({'CA': 279, 'NJ': 211, 'TX': 162, 'MA': 123, 'OH': 111, 'MI': 108, 'PA': 101,
'FL': 90, 'CT': 71, 'MN': 66, 'WI': 60, 'IN': 48, 'NY': 46, 'NC': 46, 'AL': 43, 'MO':
42, 'WA': 40, 'IL': 40, 'GA': 37, 'OK': 36, 'TN': 35, 'VA': 33, 'OR': 31, 'SC': 28,
'KY': 26, 'RI': 26, 'AR': 25, 'CO': 25, 'UT': 24, 'LA': 22, 'NH': 21, 'MS': 20, 'AZ':
20, 'IA': 20, 'ME': 17, 'WV': 14, 'MD': 12, 'NM': 10, 'SD': 9, 'ND': 8, 'WY': 7, 'I
D': 7, 'NV': 5, 'VT': 4, 'AK': 3, 'KS': 1, 'DE': 1, 'DC': 1})
```

At this point, I will go through each state, and aggregate them one by one into their own, new data frames. For each seperate state dataframe, I will check for missing values. Rather than checking for the total number, I want the column names of where these missing values are located. I will then confirm the total length/number of rows in the dataset, and then check the total number of missing values in each column to get an idea of how much information is missing in each column per state. If the ratio is not significant, I will then move forward and

impute the missing values as discussed previously, and then confirm if there are still any columns left with missing values.

Once confirmed that there are no longer any columns missing values, I will have to put these values back into the main dataset. I have filtered out the main dataset by state, so I will create a new dataframe titled crimedata_new. This new dataframe will include all the data, EXCEPT the data of the particular state I was working with. I will then append the seperate state dataframe to crimedata_new, replacing the old data of the state in which values were missing. Once again, I will confirm that all missing values were imputed as planned, by checking the count of missing values of crimedata_new, indexed by the particular state.

STATE: CA

In [ ]:
```
#Separate all rows represented by the state CA, and create a dataframe with them
CA = crimedata_reduced[crimedata_reduced['state']=='CA']
CA_df= pd.DataFrame(CA)

#Check for missing values: return names of each column that carries a missing value
CA_df.columns[CA_df.isna().any()]

#Check the total number of rows represented by the particular state
len(CA_df)

#Check total number of missing values in that column to establish the ratio of missing
CA_df['ViolentCrimesPerPop'].isna().sum()

#If the ratio of missing information is not significant, impute the missing values wit
CA_df['ViolentCrimesPerPop'].fillna(CA_df['ViolentCrimesPerPop'].mean(), inplace = Tru

#Confirm values were successfully imputed
CA_df.columns[CA_df.isna().any()]
```

Out[ ]:
```
Index([], dtype='object')
```

In [ ]:
```
#Create a dataframe where we filter out all rows represented by the particular state
ca_filtered = crimedata_reduced[crimedata_reduced['state'] != 'CA']

#Create a new dataframe which will serve as our main, fully imputed dataframe
#In this new dataframe, take the filtered dataframe and append the imputed state dataf
crimedata_new = pd.concat([ca_filtered, CA_df], ignore_index=True)

#Confirm there are no missing values in the rows of the particular state in the new ma
Counter(crimedata_new[crimedata_new['state'] == "CA"].isna().any())
```

Out[ ]:
```
Counter({False: 122})
```

I will repeat the previous steps with every single state represented. If no columns are returned with missing values, I will move onto the next state.

STATE: NJ

In [ ]:
```
NJ = crimedata_reduced[crimedata_reduced['state']=='NJ']
NJ_df= pd.DataFrame(NJ)
NJ_df.columns[NJ_df.isna().any()]
```

Out[ ]:
```
Index([], dtype='object')
```

STATE: TX

In [ ]:
```
TX = crimedata_reduced[crimedata_reduced['state']=='TX']
TX_df= pd.DataFrame(TX)
TX_df.columns[TX_df.isna().any()]

len(TX_df)
TX_df['rapes'].isna().sum()
TX_df['rapesPerPop'].isna().sum()
TX_df['arsons'].isna().sum()
TX_df['arsonsPerPop'].isna().sum()
TX_df['ViolentCrimesPerPop'].isna().sum()
TX_df['nonViolPerPop'].isna().sum()
```

Out[ ]:
```
5
```

In [ ]:
```
TX_df['rapes'].fillna(TX_df['rapes'].mean(), inplace = True)
TX_df['rapesPerPop'].fillna(TX_df['rapesPerPop'].mean(), inplace = True)
TX_df['arsons'].fillna(TX_df['arsons'].mean(), inplace = True)
TX_df['arsonsPerPop'].fillna(TX_df['arsonsPerPop'].mean(), inplace = True)
TX_df['ViolentCrimesPerPop'].fillna(TX_df['ViolentCrimesPerPop'].mean(), inplace = Tru
TX_df['nonViolPerPop'].fillna(TX_df['nonViolPerPop'].mean(), inplace = True)

TX_df.columns[TX_df.isna().any()]

tx_filtered = crimedata_reduced[crimedata_reduced['state'] != 'TX']
crimedata_new = pd.concat([tx_filtered, TX_df], ignore_index=True)

Counter(crimedata_new[crimedata_new['state'] == "TX"].isna().any())
```

Out[ ]:
```
Counter({False: 120, True: 2})
```

STATE: MA

In [ ]:
```
MA = crimedata_reduced[crimedata_reduced['state']=='MA']
MA_df= pd.DataFrame(MA)
MA_df.columns[MA_df.isna().any()]

len(MA_df)
MA_df['arsons'].isna().sum()
MA_df['arsonsPerPop'].isna().sum()
MA_df['ViolentCrimesPerPop'].isna().sum()
MA_df['nonViolPerPop'].isna().sum()
```

Out[ ]:
```
7
```

In [ ]:
```
MA_df['arsons'].fillna(MA_df['arsons'].mean(), inplace = True)
MA_df['arsonsPerPop'].fillna(MA_df['arsonsPerPop'].mean(), inplace = True)
MA_df['ViolentCrimesPerPop'].fillna(MA_df['ViolentCrimesPerPop'].mean(), inplace = Tru
MA_df['nonViolPerPop'].fillna(MA_df['nonViolPerPop'].mean(), inplace = True)

MA_df.columns[MA_df.isna().any()]

ma_filtered = crimedata_reduced[crimedata_reduced['state'] != 'MA']
```

```
crimedata_new = pd.concat([ma_filtered, MA_df], ignore_index=True)

Counter(crimedata_new[crimedata_new['state'] == "MA"].isna().any())
```

Out[ ]:
```
Counter({False: 122})
```

STATE: OH

In [ ]:
```
OH = crimedata_reduced[crimedata_reduced['state']=='OH']
OH_df= pd.DataFrame(OH)
OH_df.columns[OH_df.isna().any()]

len(OH_df)
OH_df['arsons'].isna().sum()
OH_df['arsonsPerPop'].isna().sum()
OH_df['ViolentCrimesPerPop'].isna().sum()
OH_df['nonViolPerPop'].isna().sum()
```

Out[ ]:
```
3
```

In [ ]:
```
OH_df['arsons'].fillna(OH_df['arsons'].mean(), inplace = True)
OH_df['arsonsPerPop'].fillna(OH_df['arsonsPerPop'].mean(), inplace = True)
OH_df['ViolentCrimesPerPop'].fillna(OH_df['ViolentCrimesPerPop'].mean(), inplace = Tru
OH_df['nonViolPerPop'].fillna(OH_df['nonViolPerPop'].mean(), inplace = True)

OH_df.columns[OH_df.isna().any()]

oh_filtered = crimedata_reduced[crimedata_reduced['state'] != 'OH']
crimedata_new = pd.concat([oh_filtered, OH_df], ignore_index=True)

Counter(crimedata_new[crimedata_new['state'] == "OH"].isna().any())
```

Out[ ]:
```
Counter({False: 122})
```

***For the state MI, I have run into a problem. Out of the 6 columns which include missing values, 3 of the columns are able to be imputed, as they are not missing a significant amount of values. However the other 3 are missing ALL values in the entire column. This particular state is represented by 108 rows. The columns, rapes, rapesPerPop and ViolentCrimesPerPop are missing all 108 values.

STATE: MI

In [ ]:
```
MI = crimedata_reduced[crimedata_reduced['state']=='MI']
MI_df= pd.DataFrame(MI)
MI_df.columns[MI_df.isna().any()]

len(MI_df)

#TOTAL rows for MI was 108
#The three columns beneath are missing ALL 108 values in these columns, there are no v
MI_df['ViolentCrimesPerPop'].isna().sum()
MI_df['rapes'].isna().sum()
MI_df['rapesPerPop'].isna().sum()

#The following rows are not missing an insignificant amount of values, they can be imp
MI_df['arsons'].isna().sum()
```

```
MI_df['arsonsPerPop'].isna().sum()
MI_df['nonViolPerPop'].isna().sum()
```

Out[ ]:
```
1
```

In [ ]:
```
#Impute the columns that are appropriate to be imputed by the mean

MI_df['arsons'].fillna(MI_df['arsons'].mean(), inplace = True)
MI_df['arsonsPerPop'].fillna(MI_df['arsonsPerPop'].mean(), inplace = True)
MI_df['nonViolPerPop'].fillna(MI_df['nonViolPerPop'].mean(), inplace = True)

MI_df.columns[MI_df.isna().any()]

mi_filtered = crimedata_reduced[crimedata_reduced['state'] != 'MI']
crimedata_new = pd.concat([mi_filtered, MI_df], ignore_index=True)

Counter(crimedata_new[crimedata_new['state'] == "MI"].isna().any())
MI_df.columns[MI_df.isna().any()]

#There are still 3 columns that have not been addressed
#These columns must be revisited
```

Out[ ]:
```
Index(['rapes', 'rapesPerPop', 'ViolentCrimesPerPop'], dtype='object')
```

STATE: PA

In [ ]:
```
PA = crimedata_reduced[crimedata_reduced['state']=='PA']
PA_df= pd.DataFrame(PA)
PA_df.columns[PA_df.isna().any()]

len(PA_df)
PA_df['arsons'].isna().sum()
PA_df['arsonsPerPop'].isna().sum()
PA_df['nonViolPerPop'].isna().sum()
```

Out[ ]:
```
1
```

In [ ]:
```
PA_df['arsons'].fillna(PA_df['arsons'].mean(), inplace = True)
PA_df['arsonsPerPop'].fillna(PA_df['arsonsPerPop'].mean(), inplace = True)
PA_df['nonViolPerPop'].fillna(PA_df['nonViolPerPop'].mean(), inplace = True)

PA_df.columns[PA_df.isna().any()]

pa_filtered = crimedata_reduced[crimedata_reduced['state'] != 'PA']
crimedata_new = pd.concat([pa_filtered, PA_df], ignore_index=True)

Counter(crimedata_new[crimedata_new['state'] == "PA"].isna().any())
```

Out[ ]:
```
Counter({False: 122})
```

STATE: FL

In [ ]:
```
FL = crimedata_reduced[crimedata_reduced['state']=='FL']
FL_df= pd.DataFrame(FL)
FL_df.columns[FL_df.isna().any()]
```

Out[ ]:
```
Index([], dtype='object')
```

STATE: CT

In [ ]:
```
CT = crimedata_reduced[crimedata_reduced['state']=='CT']
CT_df= pd.DataFrame(CT)
CT_df.columns[CT_df.isna().any()]

len(CT_df)
CT_df['ViolentCrimesPerPop'].isna().sum()
```

Out[ ]:
```
2
```

In [ ]:
```
CT_df['ViolentCrimesPerPop'].fillna(CT_df['ViolentCrimesPerPop'].mean(), inplace = Tru

CT_df.columns[CT_df.isna().any()]

ct_filtered = crimedata_reduced[crimedata_reduced['state'] != 'CT']
crimedata_new = pd.concat([ct_filtered, CT_df], ignore_index=True)

Counter(crimedata_new[crimedata_new['state'] == "CT"].isna().any())
```

Out[ ]:
```
Counter({False: 122})
```

STATE: MN

In [ ]:
```
MN = crimedata_reduced[crimedata_reduced['state']=='MN']
MN_df= pd.DataFrame(MN)
MN_df.columns[CA_df.isna().any()]
```

Out[ ]:
```
Index([], dtype='object')
```

STATE: WI

In [ ]:
```
WI = crimedata_reduced[crimedata_reduced['state']=='WI']
WI_df= pd.DataFrame(WI)
WI_df.columns[WI_df.isna().any()]

len(WI_df)
WI_df['arsons'].isna().sum()
WI_df['arsonsPerPop'].isna().sum()
WI_df['nonViolPerPop'].isna().sum()
```

Out[ ]:
```
1
```

In [ ]:
```
WI_df['arsons'].fillna(WI_df['arsons'].mean(), inplace = True)
WI_df['arsonsPerPop'].fillna(WI_df['arsonsPerPop'].mean(), inplace = True)
WI_df['nonViolPerPop'].fillna(WI_df['nonViolPerPop'].mean(), inplace = True)

WI_df.columns[WI_df.isna().any()]

wi_filtered = crimedata_reduced[crimedata_reduced['state'] != 'WI']
crimedata_new = pd.concat([wi_filtered, WI_df], ignore_index=True)

Counter(crimedata_new[crimedata_new['state'] == "WI"].isna().any())
```

Out[ ]:
```
Counter({False: 122})
```

STATE: IN

In [ ]:
```
IN = crimedata_reduced[crimedata_reduced['state']=='IN']
IN_df= pd.DataFrame(IN)
IN_df.columns[IN_df.isna().any()]
```

Out[ ]:
```
Index([], dtype='object')
```

STATE: NY

In [ ]:
```
NY = crimedata_reduced[crimedata_reduced['state']=='NY']
NY_df= pd.DataFrame(NY)
NY_df.columns[NY_df.isna().any()]

len(NY_df)
NY_df['arsons'].isna().sum()
NY_df['arsonsPerPop'].isna().sum()
NY_df['nonViolPerPop'].isna().sum()
```

Out[ ]:
```
17
```

In [ ]:
```
NY_df['arsons'].fillna(NY_df['arsons'].mean(), inplace = True)
NY_df['arsonsPerPop'].fillna(NY_df['arsonsPerPop'].mean(), inplace = True)
NY_df['nonViolPerPop'].fillna(NY_df['nonViolPerPop'].mean(), inplace = True)

NY_df.columns[NY_df.isna().any()]

ny_filtered = crimedata_reduced[crimedata_reduced['state'] != 'NY']
crimedata_new = pd.concat([ny_filtered, NY_df], ignore_index=True)

Counter(crimedata_new[crimedata_new['state'] == "NY"].isna().any())
```

Out[ ]:
```
Counter({False: 122})
```

STATE: NC

In [ ]:
```
NC = crimedata_reduced[crimedata_reduced['state']=='NC']
NC_df= pd.DataFrame(NC)
NC_df.columns[NC_df.isna().any()]
```

Out[ ]:
```
Index([], dtype='object')
```

STATE: AL

In [ ]:
```
AL = crimedata_reduced[crimedata_reduced['state']=='AL']
AL_df= pd.DataFrame(AL)
AL_df.columns[AL_df.isna().any()]

len(AL_df)
AL_df['arsons'].isna().sum()
AL_df['arsonsPerPop'].isna().sum()
AL_df['nonViolPerPop'].isna().sum()
```

```
len(AL_df)
```

Out[ ]:
```
43
```

In [ ]:
```
AL_df['arsons'].fillna(AL_df['arsons'].mean(), inplace = True)
AL_df['arsonsPerPop'].fillna(AL_df['arsonsPerPop'].mean(), inplace = True)
AL_df['nonViolPerPop'].fillna(AL_df['nonViolPerPop'].mean(), inplace = True)

AL_df.columns[AL_df.isna().any()]

al_filtered = crimedata_reduced[crimedata_reduced['state'] != 'AL']
crimedata_new = pd.concat([al_filtered, AL_df], ignore_index=True)

Counter(crimedata_new[crimedata_new['state'] == "AL"].isna().any())
```

Out[ ]:
```
Counter({False: 122})
```

STATE: MO

In [ ]:
```
MO = crimedata_reduced[crimedata_reduced['state']=='MO']
MO_df= pd.DataFrame(MO)
MO_df.columns[MO_df.isna().any()]
```

Out[ ]:
```
Index([], dtype='object')
```

STATE: WA

In [ ]:
```
WA = crimedata_reduced[crimedata_reduced['state']=='WA']
WA_df= pd.DataFrame(WA)
WA_df.columns[WA_df.isna().any()]

len(WA_df)
WA_df['arsons'].isna().sum()
WA_df['arsonsPerPop'].isna().sum()
WA_df['nonViolPerPop'].isna().sum()
```

Out[ ]:
```
1
```

In [ ]:
```
WA_df['arsons'].fillna(WA_df['arsons'].mean(), inplace = True)
WA_df['arsonsPerPop'].fillna(WA_df['arsonsPerPop'].mean(), inplace = True)
WA_df['nonViolPerPop'].fillna(WA_df['nonViolPerPop'].mean(), inplace = True)

WA_df.columns[WA_df.isna().any()]

wa_filtered = crimedata_reduced[crimedata_reduced['state'] != 'WA']
crimedata_new = pd.concat([wa_filtered, WA_df], ignore_index=True)

Counter(crimedata_new[crimedata_new['state'] == "WA"].isna().any())
```

Out[ ]:
```
Counter({False: 122})
```

STATE: IL **MISSING VALUES 1.0: All columns which are missing values are missing 100% of its values

In [ ]:
```
IL = crimedata_reduced[crimedata_reduced['state']=='IL']
IL_df= pd.DataFrame(IL)
IL_df.columns[IL_df.isna().any()]

len(IL_df)
IL_df['rapes'].isna().sum()
IL_df['rapesPerPop'].isna().sum()
IL_df['ViolentCrimesPerPop'].isna().sum()
```

Out[ ]:
```
40
```

STATE: GA

In [ ]:
```
GA = crimedata_reduced[crimedata_reduced['state']=='GA']
GA_df= pd.DataFrame(GA)
GA_df.columns[GA_df.isna().any()]
```

Out[ ]:
```
Index([], dtype='object')
```

STATE: OK

In [ ]:
```
OK = crimedata_reduced[crimedata_reduced['state']=='OK']
OK_df= pd.DataFrame(OK)
OK_df.columns[OK_df.isna().any()]
```

Out[ ]:
```
Index([], dtype='object')
```

STATE: TN

In [ ]:
```
TN = crimedata_reduced[crimedata_reduced['state']=='TN']
TN_df= pd.DataFrame(TN)
TN_df.columns[TN_df.isna().any()]

len(TN_df)
TN_df['arsons'].isna().sum()
TN_df['arsonsPerPop'].isna().sum()
TN_df['nonViolPerPop'].isna().sum()
```

Out[ ]:
```
2
```

In [ ]:
```
TN_df['arsons'].fillna(TN_df['arsons'].mean(), inplace = True)
TN_df['arsonsPerPop'].fillna(TN_df['arsonsPerPop'].mean(), inplace = True)
TN_df['nonViolPerPop'].fillna(TN_df['nonViolPerPop'].mean(), inplace = True)

TN_df.columns[WA_df.isna().any()]

tn_filtered = crimedata_reduced[crimedata_reduced['state'] != 'TN']
crimedata_new = pd.concat([tn_filtered, TN_df], ignore_index=True)

Counter(crimedata_new[crimedata_new['state'] == "TN"].isna().any())
```

Out[ ]:
```
Counter({False: 122})
```

STATE: VA

In [ ]:
```
VA = crimedata_reduced[crimedata_reduced['state']=='VA']
VA_df= pd.DataFrame(VA)
VA_df.columns[VA_df.isna().any()]
```

Out[ ]:
```
Index([], dtype='object')
```

STATE: OR

In [ ]:
```
OR = crimedata_reduced[crimedata_reduced['state']=='OR']
OR_df= pd.DataFrame(OR)
OR_df.columns[OR_df.isna().any()]
```

Out[ ]:
```
Index([], dtype='object')
```

STATE: SC

In [ ]:
```
SC = crimedata_reduced[crimedata_reduced['state']=='SC']
SC_df= pd.DataFrame(SC)
SC_df.columns[SC_df.isna().any()]
```

Out[ ]:
```
Index([], dtype='object')
```

STATE: KY

In [ ]:
```
KY = crimedata_reduced[crimedata_reduced['state']=='KY']
KY_df= pd.DataFrame(KY)
KY_df.columns[KY_df.isna().any()]
```

Out[ ]:
```
Index([], dtype='object')
```

STATE: RI

In [ ]:
```
RI = crimedata_reduced[crimedata_reduced['state']=='RI']
RI_df= pd.DataFrame(RI)
RI_df.columns[RI_df.isna().any()]
```

Out[ ]:
```
Index([], dtype='object')
```

STATE: AR

In [ ]:
```
AR = crimedata_reduced[crimedata_reduced['state']=='AR']
AR_df= pd.DataFrame(AR)
AR_df.columns[AR_df.isna().any()]
```

Out[ ]:
```
Index([], dtype='object')
```

STATE: CO

In [ ]:
```
CO = crimedata_reduced[crimedata_reduced['state']=='CO']
CO_df= pd.DataFrame(CO)
CO_df.columns[CO_df.isna().any()]
```

Out[ ]:
```
Index([], dtype='object')
```

STATE: UT

In [ ]:
```
UT = crimedata_reduced[crimedata_reduced['state']=='UT']
UT_df= pd.DataFrame(UT)
UT_df.columns[UT_df.isna().any()]
```

Out[ ]:
```
Index([], dtype='object')
```

STATE: LA

In [ ]:
```
LA = crimedata_reduced[crimedata_reduced['state']=='LA']
LA_df= pd.DataFrame(LA)
LA_df.columns[LA_df.isna().any()]

len(LA_df)
LA_df['arsons'].isna().sum()
LA_df['arsonsPerPop'].isna().sum()
LA_df['nonViolPerPop'].isna().sum()
```

Out[ ]:
```
3
```

In [ ]:
```
LA_df['arsons'].fillna(LA_df['arsons'].mean(), inplace = True)
LA_df['arsonsPerPop'].fillna(LA_df['arsonsPerPop'].mean(), inplace = True)
LA_df['nonViolPerPop'].fillna(LA_df['nonViolPerPop'].mean(), inplace = True)

LA_df.columns[LA_df.isna().any()]

la_filtered = crimedata_reduced[crimedata_reduced['state'] != 'LA']
crimedata_new = pd.concat([la_filtered, LA_df], ignore_index=True)

Counter(crimedata_new[crimedata_new['state'] == "LA"].isna().any())
```

Out[ ]:
```
Counter({False: 122})
```

STATE: NH

In [ ]:
```
NH = crimedata_reduced[crimedata_reduced['state']=='NH']
NH_df= pd.DataFrame(NH)
NH_df.columns[NH_df.isna().any()]
```

Out[ ]:
```
Index([], dtype='object')
```

STATE: MS

In [ ]:
```
MS = crimedata_reduced[crimedata_reduced['state']=='MS']
MS_df= pd.DataFrame(MS)
MS_df.columns[MS_df.isna().any()]

len(MS_df)
MS_df['ViolentCrimesPerPop'].isna().sum()
```

Out[ ]:
```
1
```

In [ ]:
```
MS_df['ViolentCrimesPerPop'].fillna(MS_df['ViolentCrimesPerPop'].mean(), inplace = Tru
```

```
MS_df.columns[MS_df.isna().any()]

ms_filtered = crimedata_reduced[crimedata_reduced['state'] != 'MS']
crimedata_new = pd.concat([ms_filtered, MS_df], ignore_index=True)

Counter(crimedata_new[crimedata_new['state'] == "MS"].isna().any())
```

Out[ ]:
```
Counter({False: 122})
```

STATE: AZ

In [ ]:
```
AZ = crimedata_reduced[crimedata_reduced['state']=='AZ']
AZ_df= pd.DataFrame(AZ)
AZ_df.columns[AZ_df.isna().any()]
```

Out[ ]:
```
Index([], dtype='object')
```

STATE: IA **MISSING VALUES 0.85

In [ ]:
```
IA = crimedata_reduced[crimedata_reduced['state']=='IA']
IA_df= pd.DataFrame(IA)
IA_df.columns[IA_df.isna().any()]

len(IA_df)
IA_df['arsons'].isna().sum()
IA_df['arsonsPerPop'].isna().sum()
IA_df['nonViolPerPop'].isna().sum()
```

Out[ ]:
```
17
```

STATE: ME

In [ ]:
```
ME = crimedata_reduced[crimedata_reduced['state']=='ME']
ME_df= pd.DataFrame(ME)
ME_df.columns[ME_df.isna().any()]
```

Out[ ]:
```
Index([], dtype='object')
```

STATE: WV

In [ ]:
```
WV = crimedata_reduced[crimedata_reduced['state']=='WV']
WV_df= pd.DataFrame(WV)
WV_df.columns[WV_df.isna().any()]
```

Out[ ]:
```
Index([], dtype='object')
```

STATE: MD

In [ ]:
```
MD = crimedata_reduced[crimedata_reduced['state']=='MD']
MD_df= pd.DataFrame(MD)
MD_df.columns[MD_df.isna().any()]

len(MD_df)
MD_df['arsons'].isna().sum()
```

```
MD_df['arsonsPerPop'].isna().sum()
MD_df['nonViolPerPop'].isna().sum()
```

Out[ ]:
```
3
```

In [ ]:
```
MD_df['arsons'].fillna(MD_df['arsons'].mean(), inplace = True)
MD_df['arsonsPerPop'].fillna(MD_df['arsonsPerPop'].mean(), inplace = True)
MD_df['nonViolPerPop'].fillna(MD_df['nonViolPerPop'].mean(), inplace = True)

MD_df.columns[MD_df.isna().any()]

md_filtered = crimedata_reduced[crimedata_reduced['state'] != 'MD']
crimedata_new = pd.concat([md_filtered, MD_df], ignore_index=True)

Counter(crimedata_new[crimedata_new['state'] == "MD"].isna().any())
```

Out[ ]:
```
Counter({False: 122})
```

STATE: NM

In [ ]:
```
NM = crimedata_reduced[crimedata_reduced['state']=='NM']
NM_df= pd.DataFrame(NM)
NM_df.columns[NM_df.isna().any()]
```

Out[ ]:
```
Index([], dtype='object')
```

STATE: SD

In [ ]:
```
SD = crimedata_reduced[crimedata_reduced['state']=='SD']
SD_df= pd.DataFrame(SD)
SD_df.columns[SD_df.isna().any()]

len(SD_df)
SD_df['arsons'].isna().sum()
SD_df['arsonsPerPop'].isna().sum()
SD_df['nonViolPerPop'].isna().sum()
```

Out[ ]:
```
1
```

In [ ]:
```
SD_df['arsons'].fillna(SD_df['arsons'].mean(), inplace = True)
SD_df['arsonsPerPop'].fillna(SD_df['arsonsPerPop'].mean(), inplace = True)
SD_df['nonViolPerPop'].fillna(SD_df['nonViolPerPop'].mean(), inplace = True)

SD_df.columns[SD_df.isna().any()]

sd_filtered = crimedata_reduced[crimedata_reduced['state'] != 'SD']
crimedata_new = pd.concat([sd_filtered, SD_df], ignore_index=True)

Counter(crimedata_new[crimedata_new['state'] == "SD"].isna().any())
```

Out[ ]:
```
Counter({False: 122})
```

STATE: ND

In [ ]:
```
ND = crimedata_reduced[crimedata_reduced['state']=='ND']
ND_df= pd.DataFrame(ND)
```

```
ND_df.columns[ND_df.isna().any()]
```

Out[ ]:
```
Index([], dtype='object')
```

STATE: WY

In [ ]:
```
WY = crimedata_reduced[crimedata_reduced['state']=='WY']
WY_df= pd.DataFrame(WY)
WY_df.columns[WY_df.isna().any()]
```

Out[ ]:
```
Index([], dtype='object')
```

STATE: ID

In [ ]:
```
ID = crimedata_reduced[crimedata_reduced['state']=='ID']
ID_df= pd.DataFrame(ID)
ID_df.columns[ID_df.isna().any()]
```

Out[ ]:
```
Index([], dtype='object')
```

STATE: NY

In [ ]:
```
NY = crimedata_reduced[crimedata_reduced['state']=='NY']
NY_df= pd.DataFrame(NY)
NY_df.columns[NY_df.isna().any()]

len(NY_df)
NY_df['arsons'].isna().sum()
NY_df['arsonsPerPop'].isna().sum()
NY_df['nonViolPerPop'].isna().sum()
```

Out[ ]:
```
17
```

In [ ]:
```
NY_df['arsons'].fillna(NY_df['arsons'].mean(), inplace = True)
NY_df['arsonsPerPop'].fillna(NY_df['arsonsPerPop'].mean(), inplace = True)
NY_df['nonViolPerPop'].fillna(NY_df['nonViolPerPop'].mean(), inplace = True)

NY_df.columns[NY_df.isna().any()]

ny_filtered = crimedata_reduced[crimedata_reduced['state'] != 'NY']
crimedata_new = pd.concat([ny_filtered, NY_df], ignore_index=True)

Counter(crimedata_new[crimedata_new['state'] == "NY"].isna().any())
```

Out[ ]:
```
Counter({False: 122})
```

STATE: VT **MISSING VALUES 1.0

In [ ]:
```
VT = crimedata_reduced[crimedata_reduced['state']=='VT']
VT_df= pd.DataFrame(VT)
VT_df.columns[VT_df.isna().any()]

len(VT_df)
VT_df['arsons'].isna().sum()
VT_df['arsonsPerPop'].isna().sum()
VT_df['nonViolPerPop'].isna().sum()
```

```
len(VT_df)
```

Out[ ]: 4

STATE: AK

In [ ]:
```
AK = crimedata_reduced[crimedata_reduced['state']=='AK']
AK_df= pd.DataFrame(AK)
AK_df.columns[AK_df.isna().any()]
```

Out[ ]: Index([], dtype='object')

STATE: KS *MISSING VALUES 1.0

In [ ]:
```
KS = crimedata_reduced[crimedata_reduced['state']=='KS']
KS_df= pd.DataFrame(KS)
KS_df.columns[KS_df.isna().any()]

len(KS_df)
KS_df['arsons'].isna().sum()
KS_df['arsonsPerPop'].isna().sum()
KS_df['nonViolPerPop'].isna().sum()

len(KS_df)
```

Out[ ]: 1

STATE: DE

In [ ]:
```
DE = crimedata_reduced[crimedata_reduced['state']=='DE']
DE_df= pd.DataFrame(DE)
DE_df.columns[DE_df.isna().any()]
```

Out[ ]: Index([], dtype='object')

STATE: DC

In [ ]:
```
DC = crimedata_reduced[crimedata_reduced['state']=='DC']
DC_df= pd.DataFrame(DC)
DC_df.columns[DC_df.isna().any()]
```

Out[ ]: Index([], dtype='object')

**There were a number of states that had a significant amount of missing values from some of its columns. These columns were: 'rapes', 'rapesPerPop', 'arsons', 'arsonsPerPop', 'ViolentCrimesPerPop', and 'nonViolPerPop'. The reason these were marked as significant was because they exceeded the missing value threshold of 0.65 of missing values. Some of the states are missing 100% of its values in a specific column.

Because of the nature of the amount of missing values, we are unable to impute these values with the mean. For states missing 100% of the values in a column, there is no mean to impute them with. That being said, the approach to their imputation must be handled differently. I have

considered taking the 3-4 surrounding states of this particular state, aggregate them, find the mean of this particular column and then impute the mean into the missing values. This runs the risk of introducing bias into the dataset. The other option would be to drop these rows entirely, which would include dropping 6 entire states.

These states include: MI, AL, IL, IA, VT, KS

In [ ]:
```
#Check the total number of missing values for each column that was reported to have mi

crimedata_new['rapes'].isna().sum()
crimedata_new['rapesPerPop'].isna().sum()
crimedata_new['arsons'].isna().sum()
crimedata_new['arsonsPerPop'].isna().sum()
crimedata_new['nonViolPerPop'].isna().sum()
```

Out[ ]:
```
80
```

In [ ]:
```
#DROP ALL ROWS WITH MISSING VALUES
crimedata_new = crimedata_new.dropna()
```

I have decided to drop all rows with missing values. Although we are now missing a number of entire states, the bias this may introduce (not having a fully representational sample of the United States), I believe will ultimately affect the predictive models less and introduce considerably less bias into our data as the other option of imputing the mean of surrounding areas would.

Now that we have gone through each state, imputed as necessary, and dropped rows with missing values, we will check the entire dataset for any columns with missing values that we may have missed.

In [ ]:
```
#Check whole dataset for columns with missing values
crimedata_new.columns[crimedata_new.isna().any()]

len(crimedata_new)
```

Out[ ]:
```
1919
```

In [ ]:
```
#Check datatypes again to confirm
crimedata_new.dtypes
```

Check the ranges of our target variables:

In [ ]:
```
print(crimedata_new['murdPerPop'].min())
print(crimedata_new['murdPerPop'].max())
```

```
0.0
91.09
```

In [ ]:
```
print(crimedata_new['robbbPerPop'].min())
print(crimedata_new['robbbPerPop'].max())
```

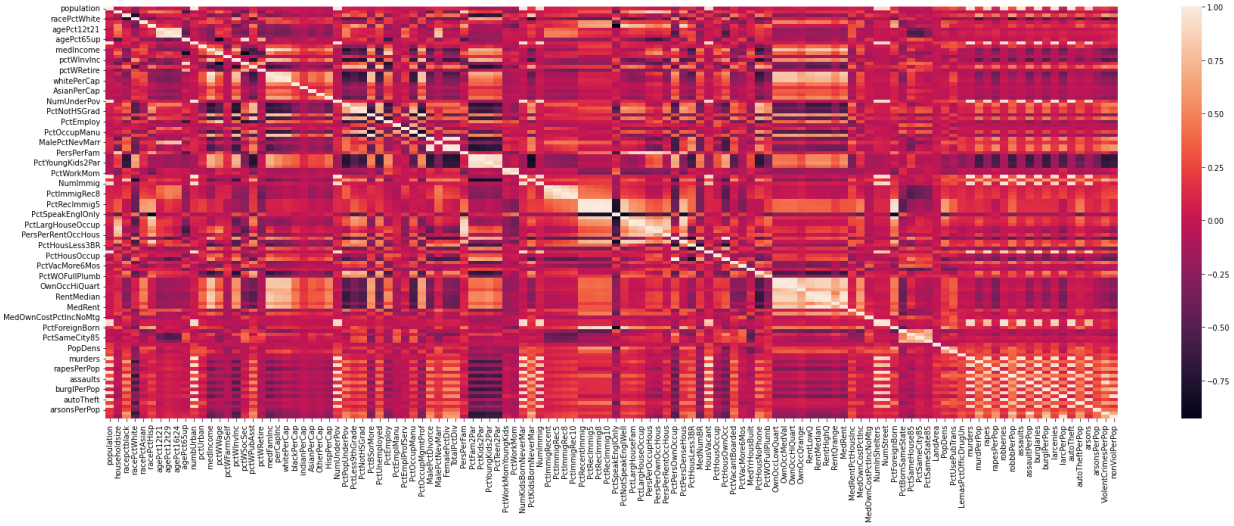
```
0.0
2264.13
```

Here we will deal with CORRELATION: I will run a correlation matrix on the entire dataset to get an idea of which attribute pairs have the highest positive and negative correlations.

In [ ]:
```
pd.set_option('display.max_rows', None)

#Run correlation matrix
crime_corr = crimedata_new.corr(numeric_only=True)
```

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

plt.figure(figsize=(30, 10))
sns.heatmap(crime_corr)
```

Out[ ]:
```
<AxesSubplot:>
```



In [ ]:
```
#Unstack the matrix and sort the values from lowest to highest so we can clearly see t
sorted_mat = crime_corr.unstack().sort_values()
sorted_mat
```

Now we will create a dataset for each target variable, giving us 2 seperate datasets. For each dataset, we will examine the correlations between each independent variable and our dependent variable, as a step of general exploration.

We will then run a full correlation matrix on the entire target variable dataset to see which attributes are highly correlated, so we can then drop the necessary columns.

MURDER CATEGORY CORRELATIONS:

In [ ]:
```
#Create dataset for the specific crime target variable

murders_data = crimedata_new.drop(['rapes', 'rapesPerPop', 'robberies', 'robbbPerPop',
                                   'burglaries', 'burglPerPop', 'larcenies', 'larcPerP
                                   'arsons', 'arsonsPerPop', 'ViolentCrimesPerPop', 'r
murders_data.head()
```

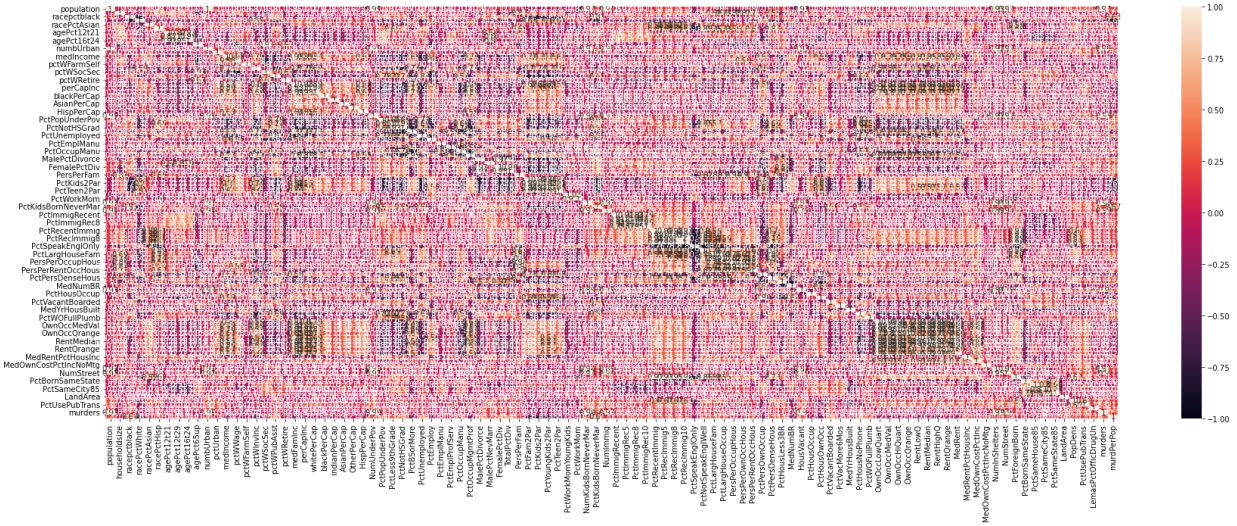
Out[ ]:

	communityname	state	population	householdsize	racepctblack	racePctWhite	racePctAsian
0	BerkeleyHeightstownship	NJ	11980	3.10	1.37	91.78	6.50
1	Marpletownship	PA	23123	2.82	0.80	95.57	3.44
2	Tigardcity	OR	29344	2.43	0.74	94.33	3.43
4	Springfieldcity	MO	140494	2.45	2.51	95.65	0.90
5	Norwoodtown	MA	28700	2.60	1.60	96.57	1.47

In [ ]:
```
for col in murders_data.columns:
    print(col)
```

In [ ]:
```
plt.figure(figsize=(30, 10))
heatmap = sns.heatmap(murders_data.corr(), vmin=-1, vmax=1, annot=True)
```

```
C:\Users\radon\AppData\Local\Temp\ipykernel_10788\312952547.py:2: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only valid columns or specify the value of numeric_only to silence this warning.
  heatmap = sns.heatmap(murders_data.corr(), vmin=-1, vmax=1, annot=True)
```



In [ ]:
```
#Examine the correlations of each column with 'murdPerPop' as our target variable
murdPerPop_corr = murders_data[murders_data.columns[1:]].corr()['murdPerPop'][:]

#Sort the values so we can see the positive and negative correlation clearly
murdPerPop_corr.sort_values()
```

In [ ]:
```
#Examine the correlations of each columns and 'murders' as our target variable
murders_corr = murders_data[murders_data.columns[1:]].corr()['murders'][:]

#Sort the values so we can see the positive and negative correlations clearly
murders_corr.sort_values()
```

In [ ]:
```
#murd_corr = murders_data.corr(method='spearman')

#Create a correlation matrix for the dataframe category
murd_corr = murders_data.corr(numeric_only=True).abs()
```

```
#Select the upper triangle of the matrix, excluding the diagnonal elements
murd_tri = murd_corr.where(np.triu(np.ones(murd_corr.shape),k=1).astype(bool))

#drop the columns with a correlation greater than 0.8 and make a list of those columns
murd_drop = [column for column in murd_tri.columns if any(murd_tri[column] > 0.8)]

#drop the murd_drop columns from the dataframe
murders = murders_data.drop(murders_data[murd_drop], axis=1)
```

In [ ]:
```
#for col in murders.columns:
    #print(col)

len(murders.columns)
```

Out[ ]:
```
53
```

ROBEERIES CATEGORY CORRELATIONS:

In [ ]:
```
robberies_data = crimedata_new.drop(['rapes', 'rapesPerPop', 'murders', 'murdPerPop',
robberies_data.head()
```

Out[ ]:

	communityname	state	population	householdsize	racepctblack	racePctWhite	racePctAsian
0	BerkeleyHeightstownship	NJ	11980	3.10	1.37	91.78	6.50
1	Marpletownship	PA	23123	2.82	0.80	95.57	3.44
2	Tigardcity	OR	29344	2.43	0.74	94.33	3.43
4	Springfieldcity	MO	140494	2.45	2.51	95.65	0.90
5	Norwoodtown	MA	28700	2.60	1.60	96.57	1.47

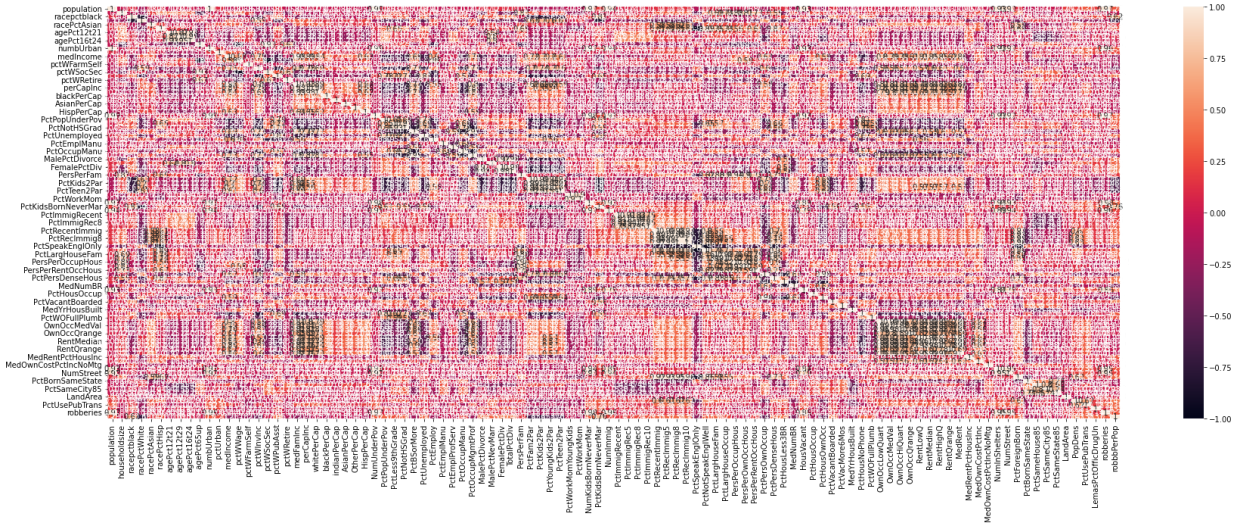
In [ ]:
```
robberies_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1919 entries, 0 to 2214
Columns: 106 entries, communityname to robbbPerPop
dtypes: float64(77), int64(27), object(2)
memory usage: 1.6+ MB
```

In [ ]:
```
plt.figure(figsize=(30, 10))
sns.heatmap(robberies_data.corr(), vmin=-1, vmax=1, annot=True)
```

```
C:\Users\radon\AppData\Local\Temp\ipykernel_10788\1798950277.py:2: FutureWarning: The
default value of numeric_only in DataFrame.corr is deprecated. In a future version, i
t will default to False. Select only valid columns or specify the value of numeric_on
ly to silence this warning.
  sns.heatmap(robberies_data.corr(), vmin=-1, vmax=1, annot=True)
```

Out[ ]:
```
<AxesSubplot:>
```



In [ ]:
```
#Examine the correlations of each columns and 'robbbPerPop' as our target variable
robbbPerPop_corr = robberies_data[robberies_data.columns[1:]].corr()['robbbPerPop'][:]

#Sort the values so we can see the positive and negative correlations clearly
robbbPerPop_corr.sort_values()
```

In [ ]:
```
#Examine the correlations of each columns and 'robberies' as our target variable
robberies_corr = robberies_data[robberies_data.columns[1:]].corr()['robberies'][:]

#Sort the values so we can see the positive and negative correlations clearly
robberies_corr.sort_values()
```

In [ ]:
```
#Create a correlation matrix for the dataframe category
robb_corr = robberies_data.corr(numeric_only=True).abs()

#Select the upper triangle of the matrix, excluding the diagnonal elements
robb_tri = robb_corr.where(np.triu(np.ones(robb_corr.shape),k=1).astype(bool))

#drop the columns with a correlation greater than 0.8 and make a list of those columns
robb_drop = [column for column in robb_tri.columns if any(robb_tri[column] > 0.8)]

#drop the columns in the previous list from the dataframe
robberies = robberies_data.drop(robberies_data[robb_drop], axis=1)

robberies.head()
```

Out[ ]:

	communityname	state	population	householdsize	racepctblack	racePctWhite	racePctAsian
0	BerkeleyHeightstownship	NJ	11980	3.10	1.37	91.78	6.50
1	Marpletownship	PA	23123	2.82	0.80	95.57	3.44
2	Tigardcity	OR	29344	2.43	0.74	94.33	3.43
4	Springfieldcity	MO	140494	2.45	2.51	95.65	0.90
5	Norwoodtown	MA	28700	2.60	1.60	96.57	1.47

In [ ]:
```
robberies[robberies.columns[1:]].corr()['robbbPerPop'][:]
```

In [ ]:
```
murders_copy[murders_copy.columns[1:]].corr()['murdPerPop'][:]
```

Here I have chosen to make a copy of the original murders dataset. I did not make a copy before altering the original dataset, so here I have backtracked in order to do this.

In [ ]:
```
#Create a correlation matrix for the dataframe category
murd_corr2 = murders_data.corr(numeric_only=True).abs()

#Select the upper triangle of the matrix, excluding the diagnonal elements
murd_tri2 = murd_corr2.where(np.triu(np.ones(murd_corr.shape),k=1).astype(bool))

#drop the columns with a correlation greater than 0.8 and make a list of those columns
murd_drop2 = [column for column in murd_tri2.columns if any(murd_tri2[column] > 0.8)]

#drop the murd_drop columns from the dataframe
murders_copy = murders_data.drop(murders_data[murd_drop2], axis=1)

murders_copy.head()
```

Out[ ]:

	communityname	state	population	householdsize	racepctblack	racePctWhite	racePctAsian
0	BerkeleyHeightstownship	NJ	11980	3.10	1.37	91.78	6.50
1	Marpletownship	PA	23123	2.82	0.80	95.57	3.44
2	Tigardcity	OR	29344	2.43	0.74	94.33	3.43
4	Springfieldcity	MO	140494	2.45	2.51	95.65	0.90
5	Norwoodtown	MA	28700	2.60	1.60	96.57	1.47

We will now go through the remaining attributes to review the column summaries, as well as the distributions in order to see how balanced they are. If they are normally or close to normally distributed, we will normalize the column with a range of 0-1. If they are not normally distributed and clearly skewed/unbalanced, the column will be turned to a categorical variable with different levels. We will turn our target variables into a CATEGORICAL ATTRIBUTE. The quartiles within the column summary will establish the levels that are chosen. Lastly, we will run one more method of feature selection after normalization, INFORMATION GAIN. We will have to decide on either ENTROPY or GINI. Then we will drop any further columns if needed. This will serve as preparation of our dataset for the regression and classification models.

***RUN INFORMATION GAIN (ENTROPY OR GINI) I have attempted to run information gain on top of feature selection via correlation. I was unable to assess the issue effectively and will not be moving forward without it for now. I may revisit this at a later time.

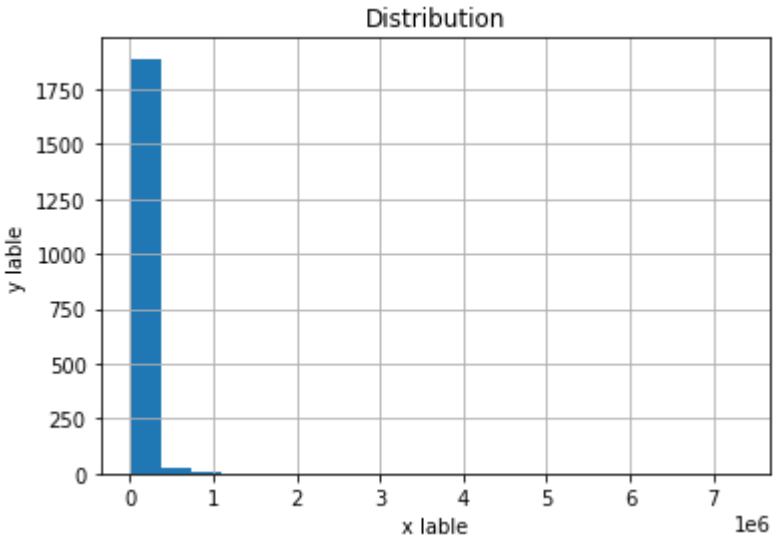
In [ ]:
```
#Examine the summaries of each attribute
murders.describe()
```

Out[ ]:

	population	householdsize	racepctblack	racePctWhite	racePctAsian	racePctHisp	agePct12t
count	1.919000e+03	1919.000000	1919.000000	1919.000000	1919.000000	1919.000000	1919.0000
mean	5.222536e+04	2.710662	9.353835	83.515268	2.804054	8.673700	14.4042
std	2.051729e+05	0.347113	13.914794	16.325927	4.720112	15.388139	4.4793
min	1.000500e+04	1.600000	0.000000	2.680000	0.060000	0.120000	4.5800
25%	1.430700e+04	2.490000	0.930000	75.885000	0.625000	0.950000	12.2100
50%	2.258000e+04	2.660000	3.070000	89.610000	1.260000	2.430000	13.6200
75%	4.308450e+04	2.855000	11.390000	95.965000	2.815000	8.910000	15.3900
max	7.322564e+06	5.280000	96.670000	99.630000	57.460000	95.290000	54.4000

In [ ]:
```
#Check the skew values of all attributes, sort them in ascending order
murders.skew(numeric_only=True).sort_values()
```

In [ ]:
```
#Find square root to establish how many bins are appropriate
import math
len(murders)
math.sqrt(2215)

#Plot the distribution
import matplotlib.pyplot as plt
murders['population'].hist(bins=20)
plt.title('Distribution')
plt.xlabel('x lable')
plt.ylabel('y lable')
plt.show()
```



In [ ]:
```
#Check the minimum and maximum value
print(murders['population'].min())
print(murders['population'].max())

#Determine bins based on quantiles
pop_bins = pd.qcut(murders['population'], q=5)
#Check the value counts of each bin to ensure they are balanced
```

```
pop_bins.value_counts()

#Create bin labels
pop_bin_labels = ['10000-13500', '13500-19000', '19000-29000', '29000-51500', '515000-
#Create bins
pop_bin = [10000, 13500, 19000, 29000, 51500, 7500000]
#Add new category
murders['pop_bins'] = pd.cut(murders['population'], bins=pop_bin, labels=pop_bin_label

murders.head()
```

```
10005
7322564
```

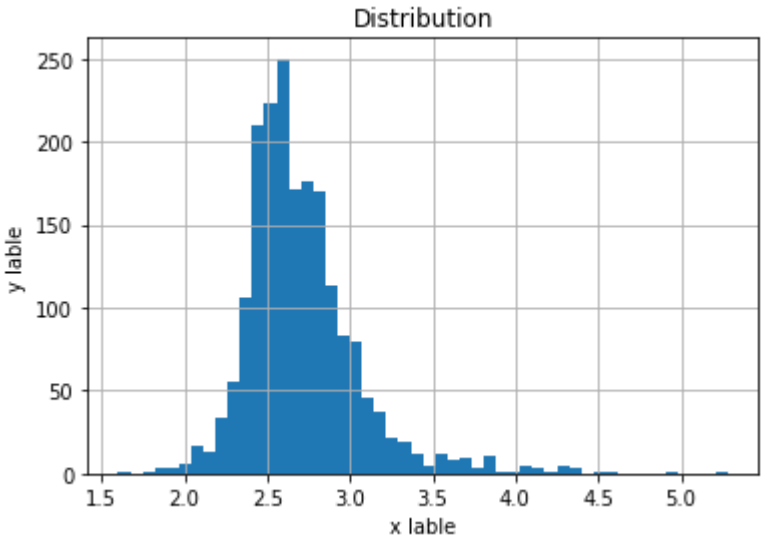
Out[ ]:

	communityname	state	population	householdsize	racepctblack	racePctWhite	racePctAsian
0	BerkeleyHeightstownship	NJ	11980	3.10	1.37	91.78	6.50
1	Marpletownship	PA	23123	2.82	0.80	95.57	3.44
2	Tigardcity	OR	29344	2.43	0.74	94.33	3.43
4	Springfieldcity	MO	140494	2.45	2.51	95.65	0.90
5	Norwoodtown	MA	28700	2.60	1.60	96.57	1.47

In [ ]:
```
murders['pop_bins'].value_counts()
```

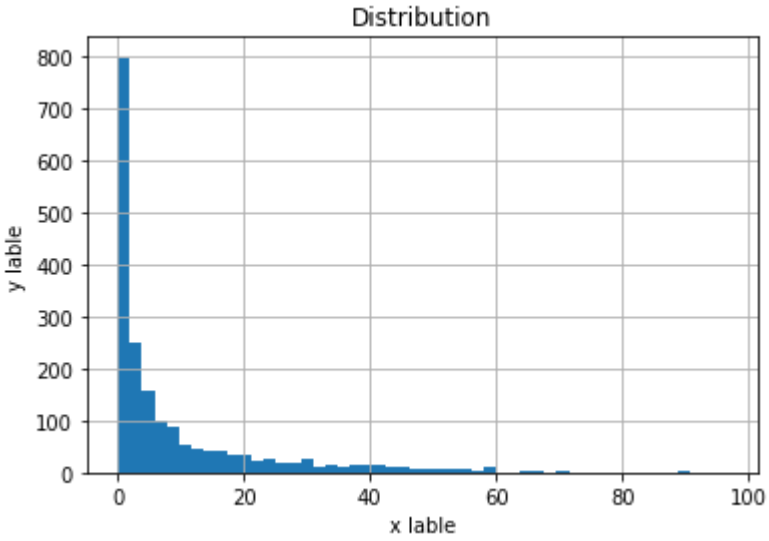
Out[ ]:
```
10000-13500       410
13500-19000       387
515000-7500000    384
19000-29000       382
29000-51500       356
Name: pop_bins, dtype: int64
```

In [ ]:
```
murders['householdsize'].hist(bins=50)
plt.title('Distribution')
plt.xlabel('x lable')
plt.ylabel('y lable')
plt.show()

#SHAPIRO TEST HERE
```



In [ ]:
```
# apply normalization technique, using a range of [0,1]
murders['householdsize_bins'] = (murders['householdsize'] - murders['householdsize'].m

murders.head()
```

In [ ]:
```
murders['racepctblack'].hist(bins=50)
plt.title('Distribution')
plt.xlabel('x lable')
plt.ylabel('y lable')
plt.show()
```



In [ ]:
```
#Check the minimum and maximum value
print(murders['racepctblack'].min())
print(murders['racepctblack'].max())

#Determine bins based on quantiles
racepctblack_bins = pd.qcut(murders['racepctblack'], q=4)
#Check the value counts of each bin to ensure they are balanced
racepctblack_bins.value_counts()

#Create bin labels
racepctblack_bin_labels = ['0-0.8%', '0.9-2.8%', '2.9-11.1%', '11.2-97%']
```

```
#Create new bin category
murders['racepctblack_bins'] = pd.qcut(murders['racepctblack'],
                                       q=4,
                                       labels=racepctblack_bin_labels)

murders.head()
```

```
0.0
96.67
```

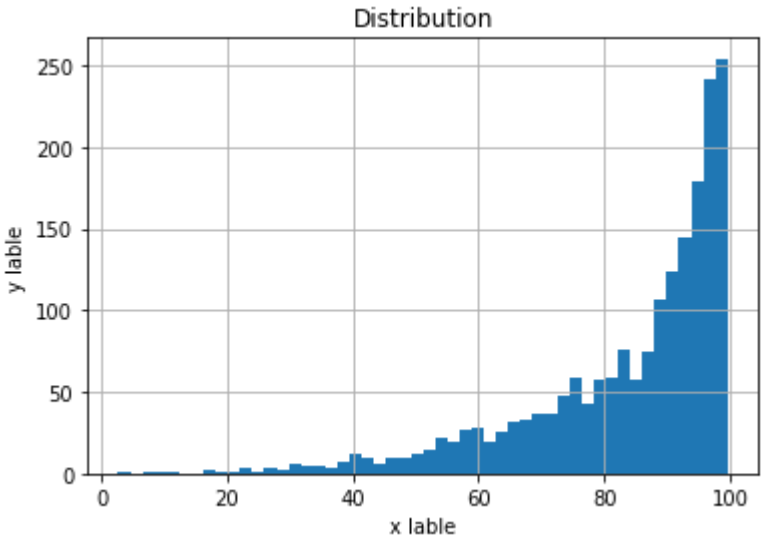
Out[ ]:

	communityname	state	population	householdsize	racepctblack	racePctWhite	racePctAsian
0	BerkeleyHeightstownship	NJ	11980	3.10	1.37	91.78	6.50
1	Marpletownship	PA	23123	2.82	0.80	95.57	3.44
2	Tigardcity	OR	29344	2.43	0.74	94.33	3.43
4	Springfieldcity	MO	140494	2.45	2.51	95.65	0.90
5	Norwoodtown	MA	28700	2.60	1.60	96.57	1.47

In [ ]:
```
murders['racepctblack_bins'].value_counts()
```

Out[ ]:
```
0-0.8%       484
11.2-97%     480
2.9-11.1%    479
0.9-2.8%     476
Name: racepctblack_bins, dtype: int64
```

In [ ]:
```
murders['racePctWhite'].hist(bins=50)
plt.title('Distribution')
plt.xlabel('x lable')
plt.ylabel('y lable')
plt.show()
```



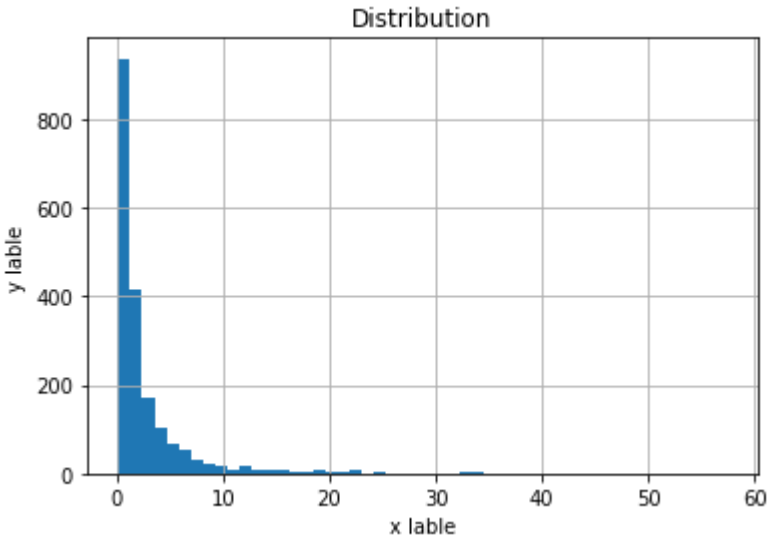
In [ ]:
```
#Check the minimum and maximum value
print(murders['racePctWhite'].min())
print(murders['racePctWhite'].max())

#Determine bins based on quantiles
racePctWhite_= pd.qcut(murders['racePctWhite'], q=4)
#Check the value counts of each bin to ensure they are balanced
racePctWhite_.value_counts()

#Create bin labels
racePctWhite_labels = ['0.0-75%','75-90%', '90-96%', '96-100%']
#Create bins
racePctWhite_bin = [0, 75, 90, 96, 100]
#Add new category
#murders_backup = murders
murders['racePctWhite_bins'] = pd.cut(murders['racePctWhite'], bins=racePctWhite_bin,

#murders.head()
```

```
2.68
99.63
```

In [ ]:
```
murders['racePctWhite_bins'].value_counts()
```

Out[ ]:
```
75-90%     518
96-100%    474
90-96%     467
0.0-75%    460
Name: racePctWhite_bins, dtype: int64
```

In [ ]:
```
murders['racePctAsian'].hist(bins=50)
plt.title('Distribution')
plt.xlabel('x lable')
plt.ylabel('y lable')
plt.show()
```



In [ ]:
```
#Check the minimum and maximum value
print(murders['racePctAsian'].min())
print(murders['racePctAsian'].max())


#Determine bins based on quantiles
```

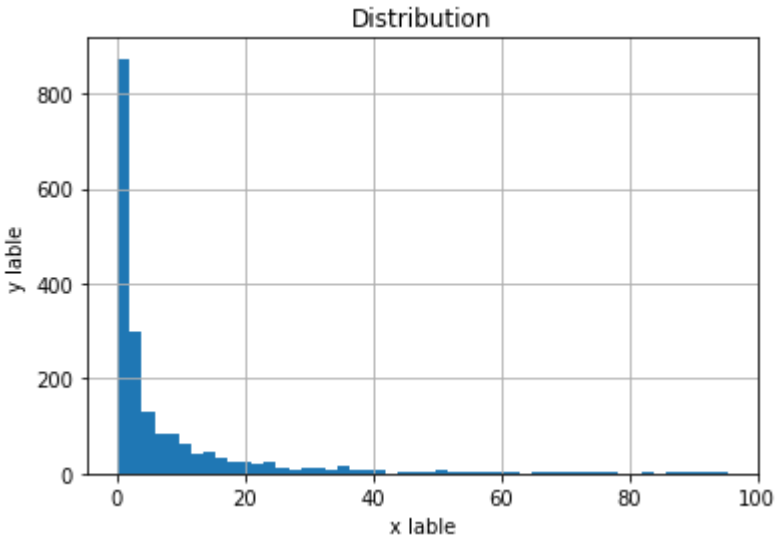
```
racePctAsian_bins = pd.qcut(murders['racePctAsian'], q=4)
#Check the value counts of each bin to ensure they are balanced
racePctAsian_bins.value_counts()

#Create bin labels
racePctAsian_bin_labels = ['0.0-0.6%', '0.6-1.2%', '1.2-2.6%', '2.7-57.5%']

#Create new bin category
murders_backup = murders
murders['racePctAsian_bins'] = pd.qcut(murders['racePctAsian'],
                                       q=4,
                                       labels=racePctAsian_bin_labels)
#murders.head()
```

```
0.06
57.46
```

In [ ]:
```
murders['racePctAsian_bins'].value_counts()
```

Out[ ]:
```
0.0-0.6%     480
0.6-1.2%     480
2.7-57.5%    480
1.2-2.6%     479
Name: racePctAsian_bins, dtype: int64
```

In [ ]:
```
murders[' racePctHisp'].hist(bins=50)
plt.title('Distribution')
plt.xlabel('x lable')
plt.ylabel('y lable')
plt.show()
```



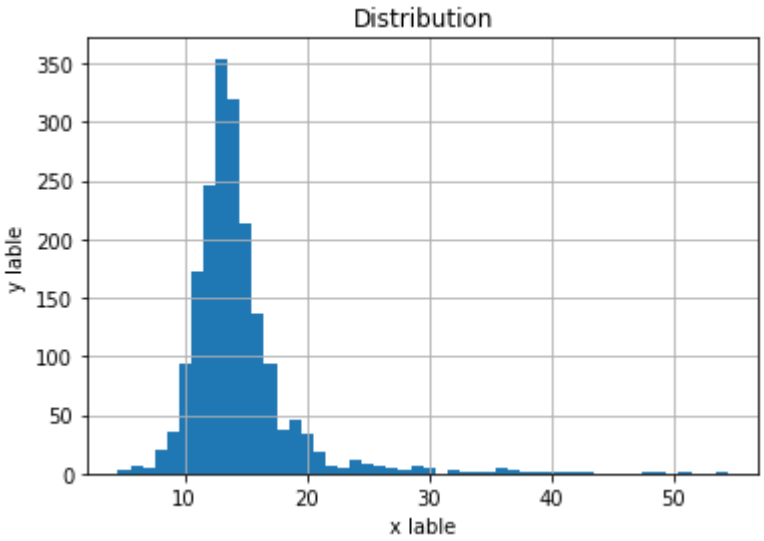
In [ ]:
```
#Check the minimum and maximum value
print(murders[' racePctHisp'].min())
print(murders[' racePctHisp'].max())

#Determine bins based on quantiles
racePctHisp_bins = pd.qcut(murders[' racePctHisp'], q=4)
#Check the value counts of each bin to ensure they are balanced
racePctHisp_bins.value_counts()

#Create bin labels
racePctHisp_bin_labels = ['0.1-0.9%', '0.9-2.2%', '2.2-7.8%', '7.8-95.3%']
```

```
#Create new bin category
murders['racePctHisp_bins'] = pd.qcut(murders[' racePctHisp'],
                                      q=4,
                                      labels=racePctHisp_bin_labels)
murders.head()
```

In [ ]:
```
murders['racePctHisp_bins'].value_counts()
```

Out[ ]:
```
0.1-0.9%     485
2.2-7.8%     480
7.8-95.3%    479
0.9-2.2%     475
Name: racePctHisp_bins, dtype: int64
```

In [ ]:
```
murders['agePct12t21'].hist(bins=50)
plt.title('Distribution')
plt.xlabel('x lable')
plt.ylabel('y lable')
plt.show()
```



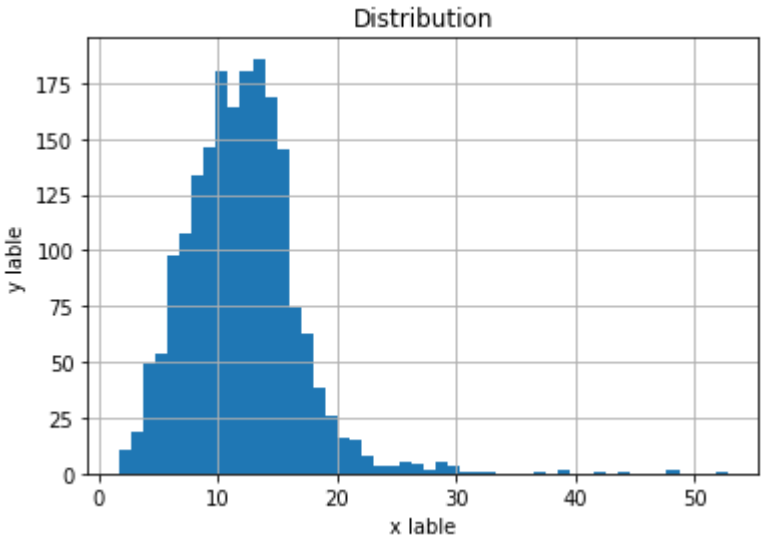
In [ ]:
```
# apply normalization technique, using a range of [0,1]
murders['agePct12t21_norm'] = (murders['agePct12t21'] - murders['agePct12t21'].min())

murders.head()
```

Out[ ]:

	communityname	state	population	householdsize	racepctblack	racePctWhite	racePctAsian
0	BerkeleyHeightstownship	NJ	11980	3.10	1.37	91.78	6.50
1	Marpletownship	PA	23123	2.82	0.80	95.57	3.44
2	Tigardcity	OR	29344	2.43	0.74	94.33	3.43
4	Springfieldcity	MO	140494	2.45	2.51	95.65	0.90
5	Norwoodtown	MA	28700	2.60	1.60	96.57	1.47

In [ ]:
```
murders = murders.drop(['householdsize_bins', 'agePct12t21_bins'], axis=1)
```

In [ ]:
```
murders[' agePct65up'].hist(bins=50)
plt.title('Distribution')
plt.xlabel('x lable')
plt.ylabel('y lable')
plt.show()
```



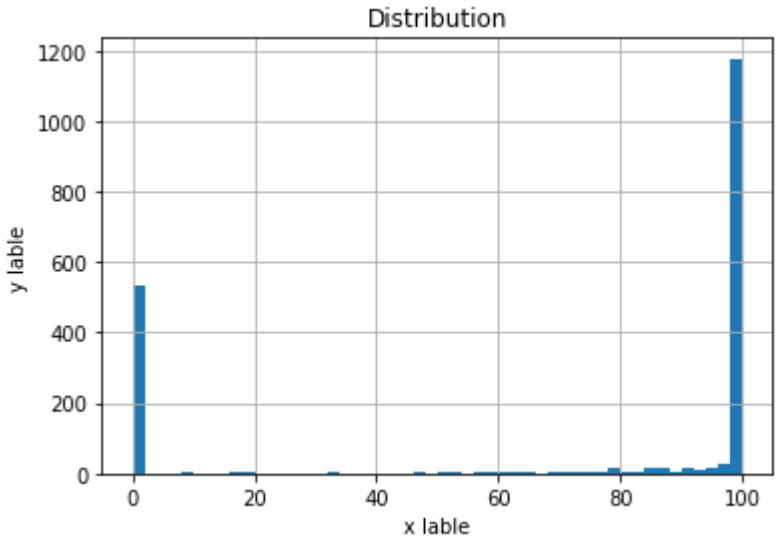
In [ ]:
```
# apply normalization technique, using a range of [0,1]
murders['agePct65up_norm'] = (murders[' agePct65up'] - murders[' agePct65up'].min()) /

murders.head()
```

Out[ ]:

	communityname	state	population	householdsize	racepctblack	racePctWhite	racePctAsian
0	BerkeleyHeightstownship	NJ	11980	3.10	1.37	91.78	6.50
1	Marpletownship	PA	23123	2.82	0.80	95.57	3.44
2	Tigardcity	OR	29344	2.43	0.74	94.33	3.43
4	Springfieldcity	MO	140494	2.45	2.51	95.65	0.90
5	Norwoodtown	MA	28700	2.60	1.60	96.57	1.47

In [ ]:
```
murders = murders.drop(['agePct65up_bins'], axis=1)
```

In [ ]:
```
murders[' pctUrban'].hist(bins=50)
plt.title('Distribution')
plt.xlabel('x lable')
plt.ylabel('y lable')
plt.show()
```



In [ ]:
```
#Check the minimum and maximum value
print(murders[' pctUrban'].min())
print(murders[' pctUrban'].max())

#Determine bins based on quantiles
print(murders[' pctUrban'].describe())

#Create bin labels
pctUrban_bin_labels = ['0.0%', '0.1-99%', '100%']
#Create bins
pctUrban_bin = [-1, 0.1, 99, 100]
#Add new category
murders_backup = murders
murders['pctUrban_bins'] = pd.cut(murders[' pctUrban'], bins=pctUrban_bin, labels=pctl
murders.head()
```

```
0.0
100.0
count    1919.000000
mean       69.758046
std        44.375875
min         0.000000
25%         0.000000
50%       100.000000
75%       100.000000
max       100.000000
Name:  pctUrban, dtype: float64
```

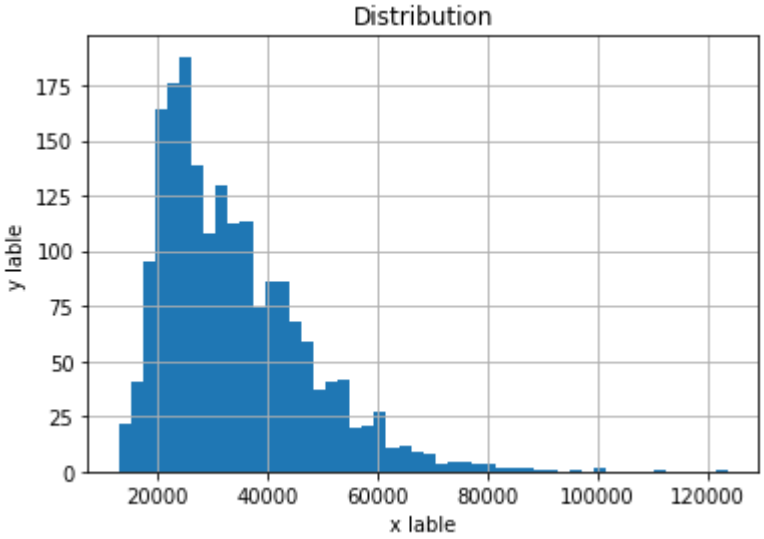
Out[ ]:

	communityname	state	population	householdsize	racepctblack	racePctWhite	racePctAsian
0	BerkeleyHeightstownship	NJ	11980	3.10	1.37	91.78	6.50
1	Marpletownship	PA	23123	2.82	0.80	95.57	3.44
2	Tigardcity	OR	29344	2.43	0.74	94.33	3.43
4	Springfieldcity	MO	140494	2.45	2.51	95.65	0.90
5	Norwoodtown	MA	28700	2.60	1.60	96.57	1.47

In [ ]:
```
murders['pctUrban_bins'].value_counts()

#This distribution was not able to be balanced in any meaningful way
#I decided to leave the values as is, changing them to a binary yes/no would have beer
```

Out[ ]:
```
100%       1158
0.0%        532
0.1-99%     229
Name: pctUrban_bins, dtype: int64
```

In [ ]:
```
murders[' medIncome'].hist(bins=50)
plt.title('Distribution')
plt.xlabel('x lable')
plt.ylabel('y lable')
plt.show()
```



In [ ]:
```
#Check the minimum and maximum value
print(murders[' medIncome'].min())
print(murders[' medIncome'].max())

#Determine bins based on quantiles
medIncome_ = pd.qcut(murders[' medIncome'], q=4)
#Check the value counts of each bin to ensure they are balanced
medIncome_.value_counts()

#Create bin labels
medIncome_labels = ['$8,000-$24,000', '$24,000-$32,000', '$32,000-$42,000', '42,000-$1
#Create bins
medIncome_bin = [8865, 24000, 32000, 42000, 123625]
#Add new category
murders_backup = murders
murders['medIncome_bins'] = pd.cut(murders[' medIncome'], bins=medIncome_bin, labels=m
murders.head()
```

```
12908
123625
```

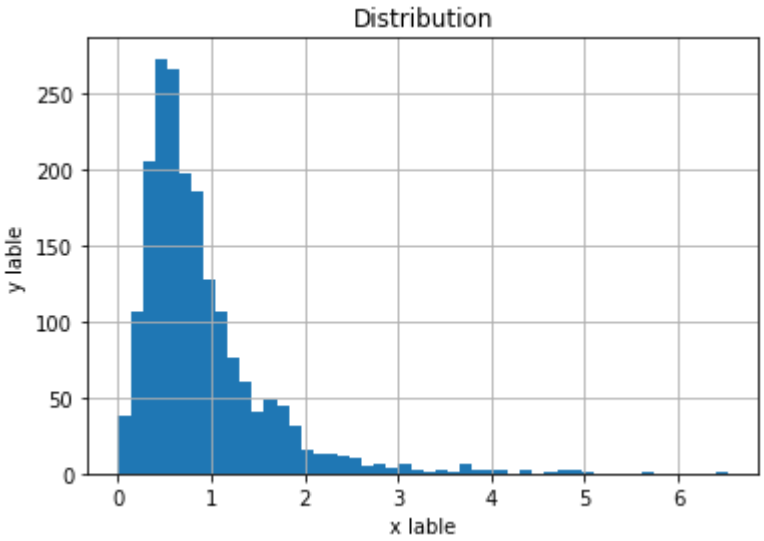
Out[ ]:

	communityname	state	population	householdsize	racepctblack	racePctWhite	racePctAsian
0	BerkeleyHeightstownship	NJ	11980	3.10	1.37	91.78	6.50
1	Marpletownship	PA	23123	2.82	0.80	95.57	3.44
2	Tigardcity	OR	29344	2.43	0.74	94.33	3.43
4	Springfieldcity	MO	140494	2.45	2.51	95.65	0.90
5	Norwoodtown	MA	28700	2.60	1.60	96.57	1.47

In [ ]:
```
murders['medIncome_bins'].value_counts()
```

Out[ ]:
```
$24,000-$32,000    511
$8,000-$24,000     500
42,000-$125,000    458
$32,000-$42,000    450
Name: medIncome_bins, dtype: int64
```

In [ ]:
```
murders['pctWFarmSelf'].hist(bins=50)
plt.title('Distribution')
plt.xlabel('x lable')
plt.ylabel('y lable')
plt.show()
```



In [ ]:
```
#Check the minimum and maximum value
print(murders['pctWFarmSelf'].min())
print(murders['pctWFarmSelf'].max())

#Determine bins based on quantiles
murders['pctWFarmSelf'].describe()

#Create bin labels
pctWFarmSelf_labels = ['0-0.5%', '0.5-0.7%', '0.7-1.0%', '1.0-7.0%']
#Create bins
pctWFarmSelf_bin = [-1, 0.5, 0.7, 1, 7]
#Add new category
murders['pctWFarmSelf_bins'] = pd.cut(murders['pctWFarmSelf'], bins=pctWFarmSelf_bin,
murders.head()
```

```
0.0
6.53
```

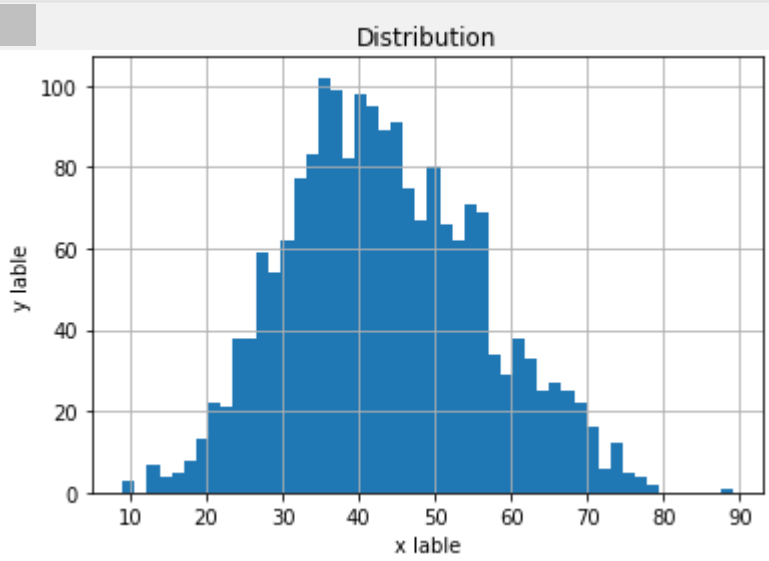
Out[ ]:

	communityname	state	population	householdsize	racepctblack	racePctWhite	racePctAsian
0	BerkeleyHeightstownship	NJ	11980	3.10	1.37	91.78	6.50
1	Marpletownship	PA	23123	2.82	0.80	95.57	3.44
2	Tigardcity	OR	29344	2.43	0.74	94.33	3.43
4	Springfieldcity	MO	140494	2.45	2.51	95.65	0.90
5	Norwoodtown	MA	28700	2.60	1.60	96.57	1.47

In [ ]:
```
murders['pctWFarmSelf_bins'].value_counts()
```

Out[ ]:
```
0-0.5%      578
1.0-7.0%    553
0.5-0.7%    401
0.7-1.0%    387
Name: pctWFarmSelf_bins, dtype: int64
```

In [ ]:
```
murders['pctWInvInc'].hist(bins=50)
plt.title('Distribution')
plt.xlabel('x lable')
plt.ylabel('y lable')
plt.show()
```

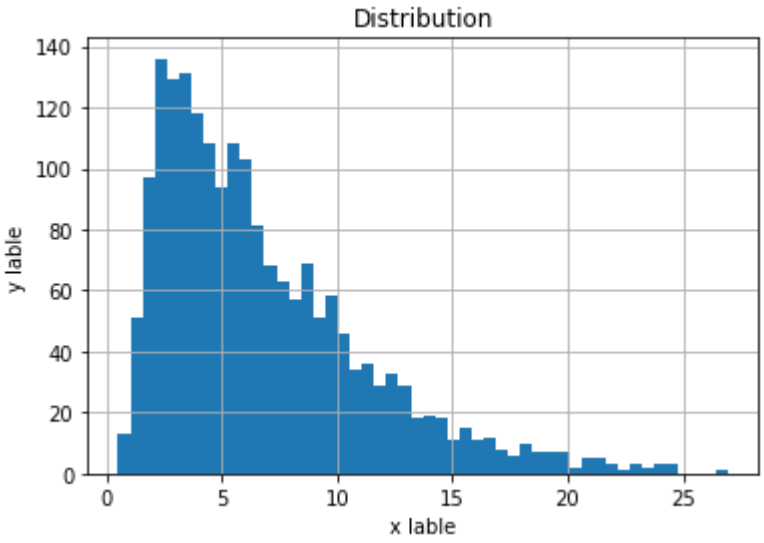


In [ ]:
```
# apply normalization technique, using a range of [0,1]
murders['pctWInvInc_norm'] = (murders['pctWInvInc'] - murders['pctWInvInc'].min()) / (

murders.head()
```

Out[ ]:

	communityname	state	population	householdsize	racepctblack	racePctWhite	racePctAsian
0	BerkeleyHeightstownship	NJ	11980	3.10	1.37	91.78	6.50
1	Marpletownship	PA	23123	2.82	0.80	95.57	3.44
2	Tigardcity	OR	29344	2.43	0.74	94.33	3.43
4	Springfieldcity	MO	140494	2.45	2.51	95.65	0.90
5	Norwoodtown	MA	28700	2.60	1.60	96.57	1.47

In [ ]:
```
murders[' pctWPubAsst'].hist(bins=50)
plt.title('Distribution')
plt.xlabel('x lable')
plt.ylabel('y lable')
plt.show()
```



In [ ]:
```
#Check the minimum and maximum value
print(murders[' pctWPubAsst'].min())
print(murders[' pctWPubAsst'].max())

#Determine bins based on quantiles
pctWPubAsst_ = pd.qcut(murders[' pctWPubAsst'], q=4)
#Check the value counts of each bin to ensure they are balanced
pctWPubAsst_.value_counts()

#Create bin labels
pctWPubAsst_labels = ['0-3%', '3-5%', '5-8%', '8-45%']
#Create bins
pctWPubAsst_bin = [0, 3, 5, 8.5, 45]
#Add new category
murders['pctWPubAsst_bins'] = pd.cut(murders[' pctWPubAsst'], bins=pctWPubAsst_bin, la
murders.head()
```

```
0.5
26.92
```

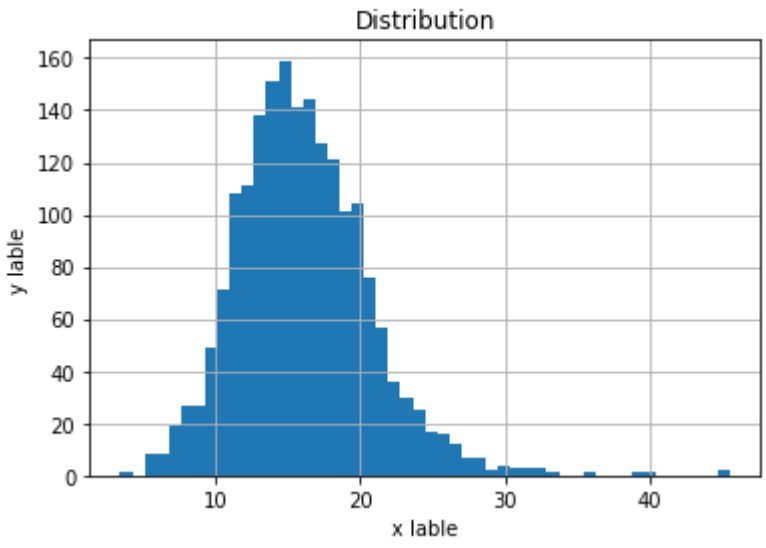
Out[ ]:

	communityname	state	population	householdsize	racepctblack	racePctWhite	racePctAsian
0	BerkeleyHeightstownship	NJ	11980	3.10	1.37	91.78	6.50
1	Marpletownship	PA	23123	2.82	0.80	95.57	3.44
2	Tigardcity	OR	29344	2.43	0.74	94.33	3.43
4	Springfieldcity	MO	140494	2.45	2.51	95.65	0.90
5	Norwoodtown	MA	28700	2.60	1.60	96.57	1.47

In [ ]:
```
murders['pctWPubAsst_bins'].value_counts()
```

Out[ ]:
```
8-45%    552
5-8%     532
3-5%     441
0-3%     394
Name: pctWPubAsst_bins, dtype: int64
```

In [ ]:
```
murders['pctWRetire'].hist(bins=50)
plt.title('Distribution')
plt.xlabel('x lable')
plt.ylabel('y lable')
plt.show()
```

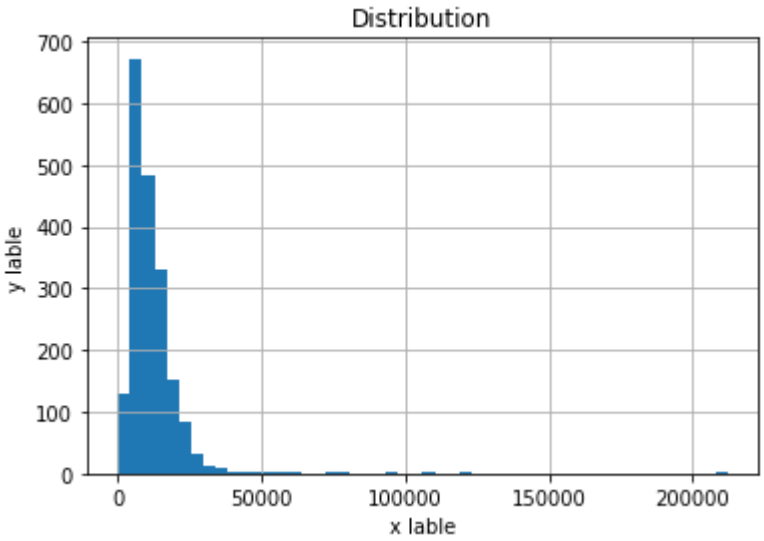


In [ ]:
```
# apply normalization technique, using a range of [0,1]
murders['pctWRetire_norm'] = (murders['pctWRetire'] - murders['pctWRetire'].min()) / (

murders.head()
```

Out[ ]:

	communityname	state	population	householdsize	racepctblack	racePctWhite	racePctAsian
0	BerkeleyHeightstownship	NJ	11980	3.10	1.37	91.78	6.50
1	Marpletownship	PA	23123	2.82	0.80	95.57	3.44
2	Tigardcity	OR	29344	2.43	0.74	94.33	3.43
4	Springfieldcity	MO	140494	2.45	2.51	95.65	0.90
5	Norwoodtown	MA	28700	2.60	1.60	96.57	1.47

In [ ]:
```
murders['blackPerCap'].hist(bins=50)
plt.title('Distribution')
plt.xlabel('x lable')
plt.ylabel('y lable')
plt.show()
```



In [ ]:
```
#Check the minimum and maximum value
print(murders['blackPerCap'].min())
print(murders['blackPerCap'].max())

#Determine bins based on quantiles
blackPerCap_ = pd.qcut(murders['blackPerCap'], q=4)
#Check the value counts of each bin to ensure they are balanced
blackPerCap_.value_counts()

#Create bin labels
blackPerCap_labels = ['0-7000', '6500-10000', '10000-15000', '15000-250000']
#Create bins
blackPerCap_bin = [-1, 7000, 10000, 15000, 250000]
#Add new catrgory
murders['blackPerCap_bins'] = pd.cut(murders['blackPerCap'], bins=blackPerCap_bin, lab
murders.head()
```

```
0
212120
```

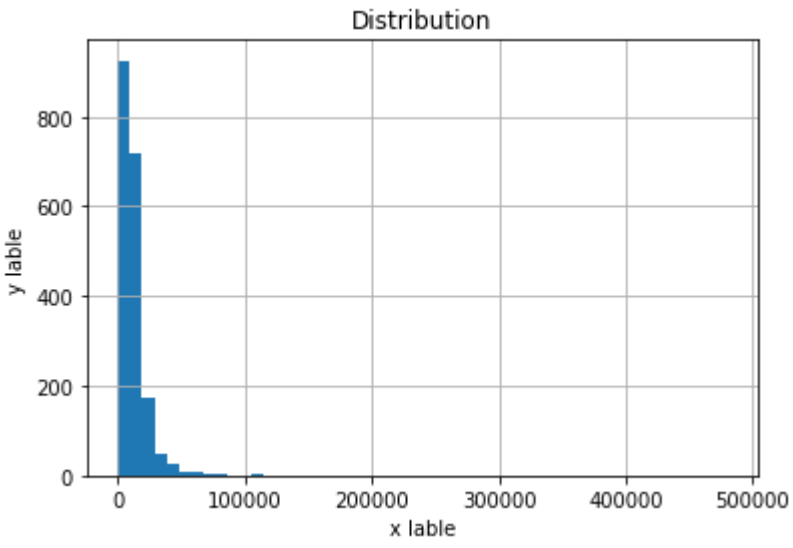
Out[ ]:

	communityname	state	population	householdsize	racepctblack	racePctWhite	racePctAsian
0	BerkeleyHeightstownship	NJ	11980	3.10	1.37	91.78	6.50
1	Marpletownship	PA	23123	2.82	0.80	95.57	3.44
2	Tigardcity	OR	29344	2.43	0.74	94.33	3.43
4	Springfieldcity	MO	140494	2.45	2.51	95.65	0.90
5	Norwoodtown	MA	28700	2.60	1.60	96.57	1.47

In [ ]:
```
murders['blackPerCap_bins'].value_counts()
```

Out[ ]:
```
0-7000          536
10000-15000     498
6500-10000      449
15000-250000    436
Name: blackPerCap_bins, dtype: int64
```

In [ ]:
```
murders['indianPerCap'].hist(bins=50)
plt.title('Distribution')
plt.xlabel('x lable')
plt.ylabel('y lable')
plt.show()
```



In [ ]:
```
#Check the minimum and maximum value
print(murders['indianPerCap'].min())
print(murders['indianPerCap'].max())

#Determine bins based on quantiles
indianPerCap_ = pd.qcut(murders['indianPerCap'], q=4)
#Check the value counts of each bin to ensure they are balanced
```

```
indianPerCap_.value_counts()

#Create bin labels
indianPerCap_labels = ['0-6500', '6500-10000', '10000-15000', '15000-500000']
#Create bins
indianPerCap_bin = [-1, 6500, 10000, 15000, 500000]
#Add new category
murders['indianPerCap_bins'] = pd.cut(murders['indianPerCap'], bins=indianPerCap_bin,
murders.head()
```

```
0
480000
```

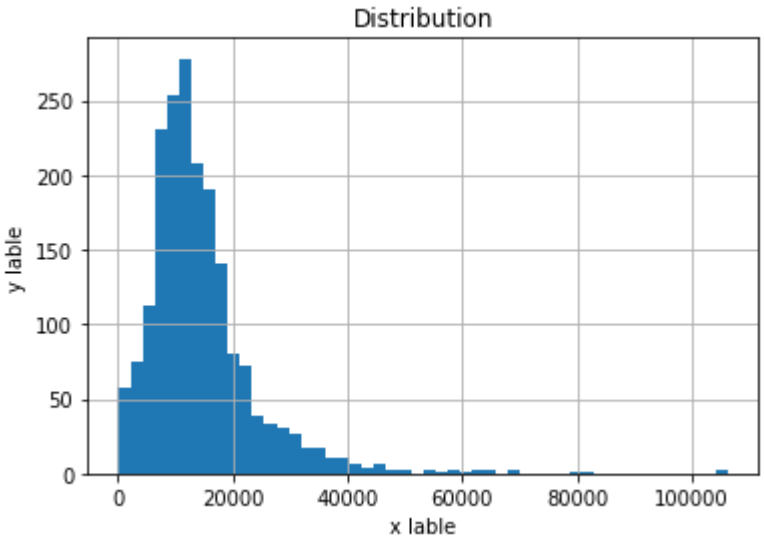
Out[ ]:

	communityname	state	population	householdsize	racepctblack	racePctWhite	racePctAsian
0	BerkeleyHeightstownship	NJ	11980	3.10	1.37	91.78	6.50
1	Marpletownship	PA	23123	2.82	0.80	95.57	3.44
2	Tigardcity	OR	29344	2.43	0.74	94.33	3.43
4	Springfieldcity	MO	140494	2.45	2.51	95.65	0.90
5	Norwoodtown	MA	28700	2.60	1.60	96.57	1.47

In [ ]:
```
murders['indianPerCap_bins'].value_counts()
```

Out[ ]:
```
0-6500          497
10000-15000     481
6500-10000      474
15000-500000    467
Name: indianPerCap_bins, dtype: int64
```

In [ ]:
```
murders['AsianPerCap'].hist(bins=50)
plt.title('Distribution')
plt.xlabel('x lable')
plt.ylabel('y lable')
plt.show()
```



In [ ]:
```
#Check the minimum and maximum value
print(murders['AsianPerCap'].min())
print(murders['AsianPerCap'].max())

#Determine bins based on quantiles
AsianPerCap_ = pd.qcut(murders['AsianPerCap'], q=4)
#Check the value counts of each bin to ensure they are balanced
AsianPerCap_.value_counts()

#Create bin labels
AsianPerCap_labels = ['0-8500', '8500-12500', '12500-17500', '17500-106500']
#Create bins
AsianPerCap_bin = [-1, 8500, 12500, 17500, 106500]
#Add new category
murders['AsianPerCap_bins'] = pd.cut(murders['AsianPerCap'], bins=AsianPerCap_bin, lab
murders.head()
```

```
0
106165
```

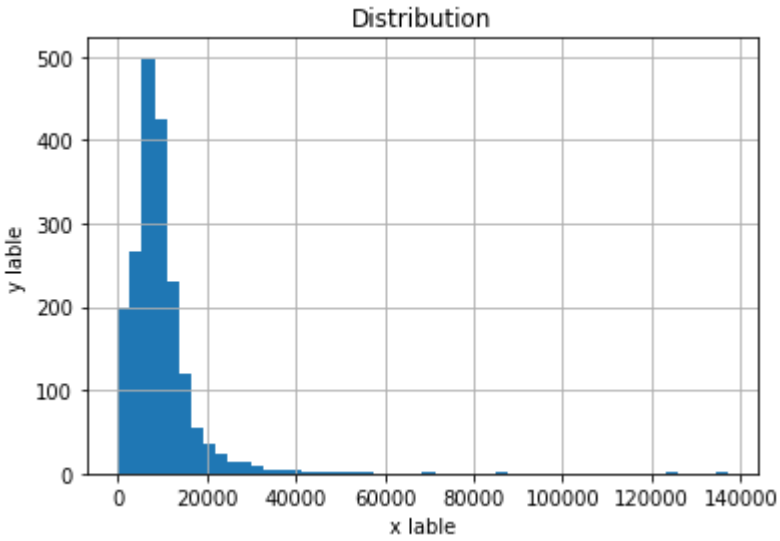
Out[ ]:

	communityname	state	population	householdsize	racepctblack	racePctWhite	racePctAsian
0	BerkeleyHeightstownship	NJ	11980	3.10	1.37	91.78	6.50
1	Marpletownship	PA	23123	2.82	0.80	95.57	3.44
2	Tigardcity	OR	29344	2.43	0.74	94.33	3.43
4	Springfieldcity	MO	140494	2.45	2.51	95.65	0.90
5	Norwoodtown	MA	28700	2.60	1.60	96.57	1.47

In [ ]:
```
murders['AsianPerCap_bins'].value_counts()
```

Out[ ]:
```
8500-12500      501
17500-106500    477
0-8500          476
12500-17500     465
Name: AsianPerCap_bins, dtype: int64
```

In [ ]:
```
murders['OtherPerCap'].hist(bins=50)
plt.title('Distribution')
plt.xlabel('x lable')
plt.ylabel('y lable')
plt.show()
```



In [ ]:
```
#Check the minimum and maximum value
print(murders['OtherPerCap'].min())
print(murders['OtherPerCap'].max())

#Determine bins based on quantiles
OtherPerCap_ = pd.qcut(murders['OtherPerCap'], q=4)
#Check the value counts of each bin to ensure they are balanced
OtherPerCap_.value_counts()

#Create bin labels
OtherPerCap_labels = ['0-5500', '5500-8000', '8000-11500', '11500-137000']
#Create bins
OtherPerCap_bin = [-1, 5500, 8000, 11500, 137000]
#Add new category
murders['OtherPerCap_bins'] = pd.cut(murders['OtherPerCap'], bins=OtherPerCap_bin, lab
murders.head()
```

```
0.0
137000.0
```

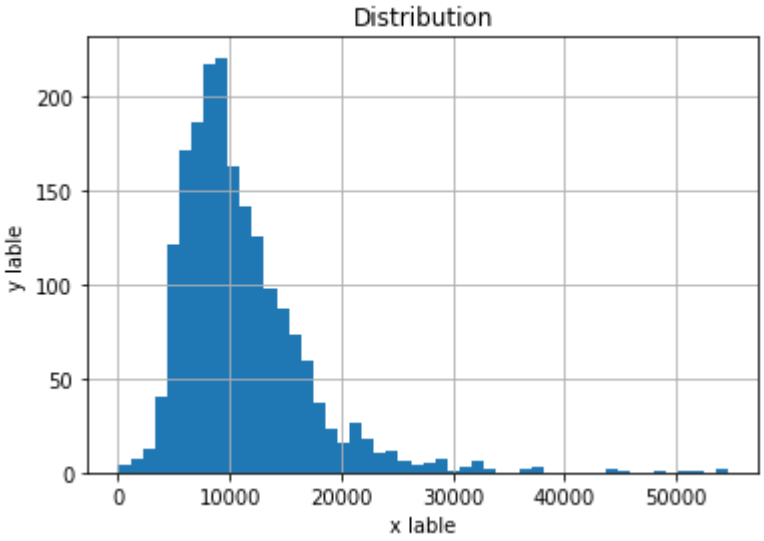
Out[ ]:

	communityname	state	population	householdsize	racepctblack	racePctWhite	racePctAsian
0	BerkeleyHeightstownship	NJ	11980	3.10	1.37	91.78	6.50
1	Marpletownship	PA	23123	2.82	0.80	95.57	3.44
2	Tigardcity	OR	29344	2.43	0.74	94.33	3.43
4	Springfieldcity	MO	140494	2.45	2.51	95.65	0.90
5	Norwoodtown	MA	28700	2.60	1.60	96.57	1.47

In [ ]:
```
murders['OtherPerCap_bins'].value_counts()
```

Out[ ]:
```
8000-11500      531
11500-137000    472
0-5500          469
5500-8000       447
Name: OtherPerCap_bins, dtype: int64
```

In [ ]:
```
murders['HispPerCap'].hist(bins=50)
plt.title('Distribution')
plt.xlabel('x lable')
plt.ylabel('y lable')
plt.show()
```



In [ ]:
```
#Check the minimum and maximum value
print(murders['HispPerCap'].min())
print(murders['HispPerCap'].max())

#Determine bins based on quantiles
HispPerCap_ = pd.qcut(murders['HispPerCap'], q=4)
#Check the value counts of each bin to ensure they are balanced
HispPerCap_.value_counts()

#Create bin labels
HispPerCap_labels = ['0-7500', '7500-1000', '1000-13500', '13500-55000']
```

```
#Create bins
HispPerCap_bin = [-1, 7500, 10000, 13500, 55000]
#Add new category
murders['HispPerCap_bins'] = pd.cut(murders['HispPerCap'], bins=HispPerCap_bin, labels
murders.head()
```

```
0
54648
```

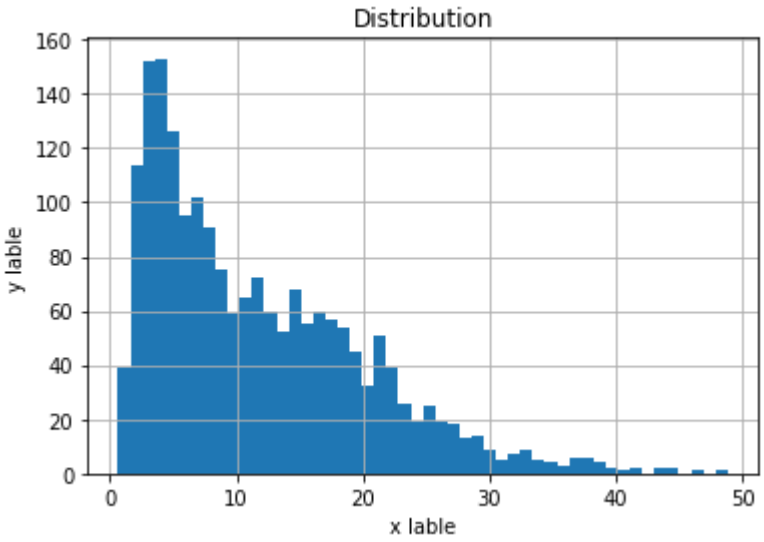
Out[ ]:

	communityname	state	population	householdsize	racepctblack	racePctWhite	racePctAsian
0	BerkeleyHeightstownship	NJ	11980	3.10	1.37	91.78	6.50
1	Marpletownship	PA	23123	2.82	0.80	95.57	3.44
2	Tigardcity	OR	29344	2.43	0.74	94.33	3.43
4	Springfieldcity	MO	140494	2.45	2.51	95.65	0.90
5	Norwoodtown	MA	28700	2.60	1.60	96.57	1.47

In [ ]:
```
murders['HispPerCap_bins'].value_counts()
```

Out[ ]:
```
0-7500          522
7500-1000       487
13500-55000     469
1000-13500      441
Name: HispPerCap_bins, dtype: int64
```

In [ ]:
```
murders['PctPopUnderPov'].hist(bins=50)
plt.title('Distribution')
plt.xlabel('x lable')
plt.ylabel('y lable')
plt.show()
```



In [ ]:
```
#Check the minimum and maximum value
print(murders['PctPopUnderPov'].min())
```

```
print(murders['PctPopUnderPov'].max())

#Determine bins based on quantiles
PctPopUnderPov_ = pd.qcut(murders['PctPopUnderPov'], q=4)
#Check the value counts of each bin to ensure they are balanced
PctPopUnderPov_.value_counts()

#Create bin labels
PctPopUnderPov_labels = ['0-5%', '5-10%', '10-17%', '17-60%']
#Create bins
PctPopUnderPov_bin = [0, 5, 10, 17, 60]
#Add new category
murders['PctPopUnderPov_bins'] = pd.cut(murders['PctPopUnderPov'], bins=PctPopUnderPov
murders.head()
```

```
0.64
48.82
```

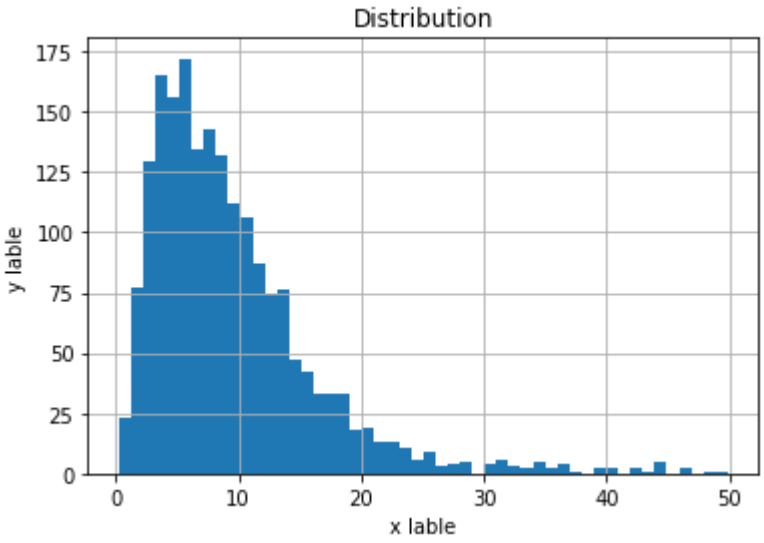
Out[ ]:

	communityname	state	population	householdsize	racepctblack	racePctWhite	racePctAsian
0	BerkeleyHeightstownship	NJ	11980	3.10	1.37	91.78	6.50
1	Marpletownship	PA	23123	2.82	0.80	95.57	3.44
2	Tigardcity	OR	29344	2.43	0.74	94.33	3.43
4	Springfieldcity	MO	140494	2.45	2.51	95.65	0.90
5	Norwoodtown	MA	28700	2.60	1.60	96.57	1.47

In [ ]:
```
murders['PctPopUnderPov_bins'].value_counts()
```

Out[ ]:
```
0-5%      522
17-60%    483
5-10%     477
10-17%    437
Name: PctPopUnderPov_bins, dtype: int64
```

In [ ]:
```
murders['PctLess9thGrade'].hist(bins=50)
plt.title('Distribution')
plt.xlabel('x lable')
plt.ylabel('y lable')
plt.show()
```



In [ ]:
```
#Check the minimum and maximum value
print(murders['PctLess9thGrade'].min())
print(murders['PctLess9thGrade'].max())

#Determine bins based on quantiles
PctLess9thGrade_ = pd.qcut(murders['PctLess9thGrade'], q=4)
#Check the value counts of each bin to ensure they are balanced
PctLess9thGrade_.value_counts()

#Create bin labels
PctLess9thGrade_labels = ['0-5%', '5-8%', '8-12%', '12-50%']
#Create bins
PctLess9thGrade_bin = [0, 5, 8, 12, 50]
#Add new category
murders['PctLess9thGrade_bins'] = pd.cut(murders['PctLess9thGrade'], bins=PctLess9thGr
murders.head()
```

```
0.2
49.89
```

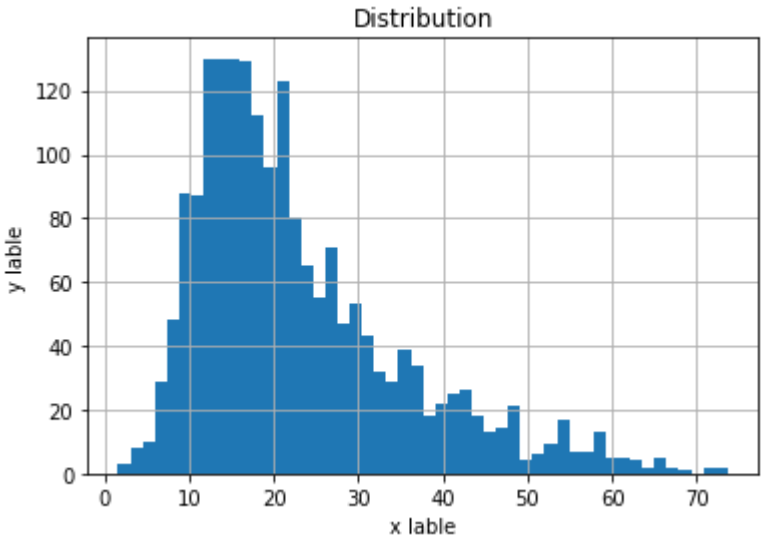
Out[ ]:

	communityname	state	population	householdsize	racepctblack	racePctWhite	racePctAsian
0	BerkeleyHeightstownship	NJ	11980	3.10	1.37	91.78	6.50
1	Marpletownship	PA	23123	2.82	0.80	95.57	3.44
2	Tigardcity	OR	29344	2.43	0.74	94.33	3.43
4	Springfieldcity	MO	140494	2.45	2.51	95.65	0.90
5	Norwoodtown	MA	28700	2.60	1.60	96.57	1.47

In [ ]:
```
murders['PctLess9thGrade_bins'].value_counts()
```

Out[ ]:
```
0-5%      516
12-50%    488
5-8%      462
8-12%     453
Name: PctLess9thGrade_bins, dtype: int64
```

In [ ]:
```
murders['PctBSorMore'].hist(bins=50)
plt.title('Distribution')
plt.xlabel('x lable')
plt.ylabel('y lable')
plt.show()
```



In [ ]:
```
#Check the minimum and maximum value
print(murders['PctBSorMore'].min())
print(murders['PctBSorMore'].max())

#Determine bins based on quantiles
PctBSorMore_ = pd.qcut(murders['PctBSorMore'], q=4)
#Check the value counts of each bin to ensure they are balanced
PctBSorMore_.value_counts()

#Create bin labels
PctBSorMore_labels = ['0-14', '14-19', '19-30', '30-80']
#Create bins
PctBSorMore_bin = [0, 14, 19, 30, 80]
#Add new category
murders['PctBSorMore_bins'] = pd.cut(murders['PctBSorMore'], bins=PctBSorMore_bin, lab
murders.head()
```

```
1.63
73.63
```

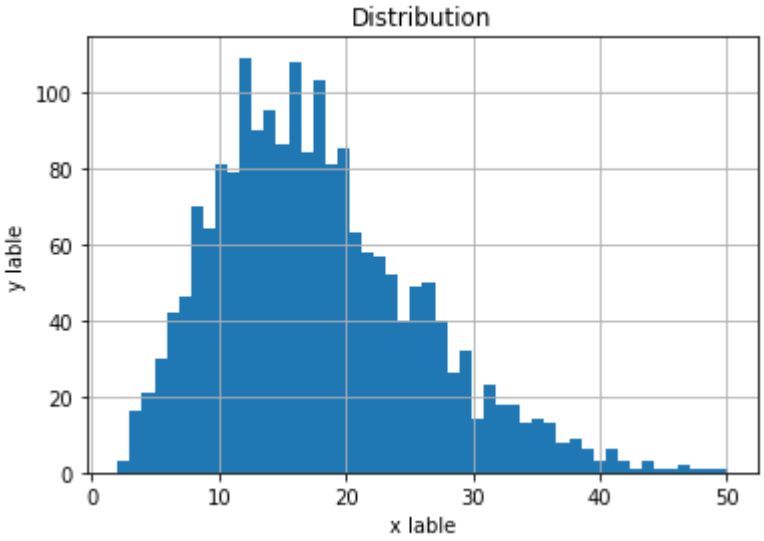
Out[ ]:

	communityname	state	population	householdsize	racepctblack	racePctWhite	racePctAsian
0	BerkeleyHeightstownship	NJ	11980	3.10	1.37	91.78	6.50
1	Marpletownship	PA	23123	2.82	0.80	95.57	3.44
2	Tigardcity	OR	29344	2.43	0.74	94.33	3.43
4	Springfieldcity	MO	140494	2.45	2.51	95.65	0.90
5	Norwoodtown	MA	28700	2.60	1.60	96.57	1.47

In [ ]:
```
murders['PctBSorMore_bins'].value_counts()
```

Out[ ]:
```
19-30    565
0-14     473
14-19    443
30-80    438
Name: PctBSorMore_bins, dtype: int64
```

In [ ]:
```
murders['PctEmplManu'].hist(bins=50)
plt.title('Distribution')
plt.xlabel('x lable')
plt.ylabel('y lable')
plt.show()
```

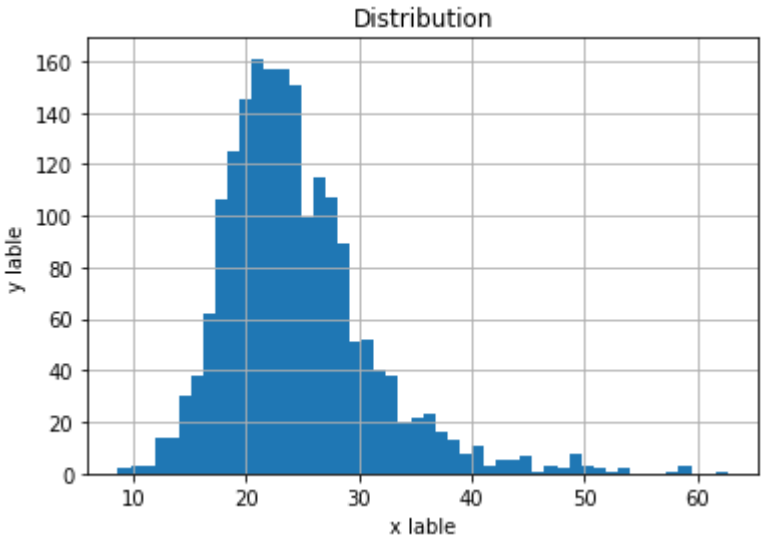


In [ ]:
```
# apply normalization technique, using a range of [0,1]
murders['PctEmplManu_norm'] = (murders['PctEmplManu'] - murders['PctEmplManu'].min())

murders.head()
```

Out[ ]:

	communityname	state	population	householdsize	racepctblack	racePctWhite	racePctAsian
0	BerkeleyHeightstownship	NJ	11980	3.10	1.37	91.78	6.50
1	Marpletownship	PA	23123	2.82	0.80	95.57	3.44
2	Tigardcity	OR	29344	2.43	0.74	94.33	3.43
4	Springfieldcity	MO	140494	2.45	2.51	95.65	0.90
5	Norwoodtown	MA	28700	2.60	1.60	96.57	1.47

In [ ]:
```
murders['PctEmplProfServ'].hist(bins=50)
plt.title('Distribution')
plt.xlabel('x lable')
plt.ylabel('y lable')
plt.show()
```



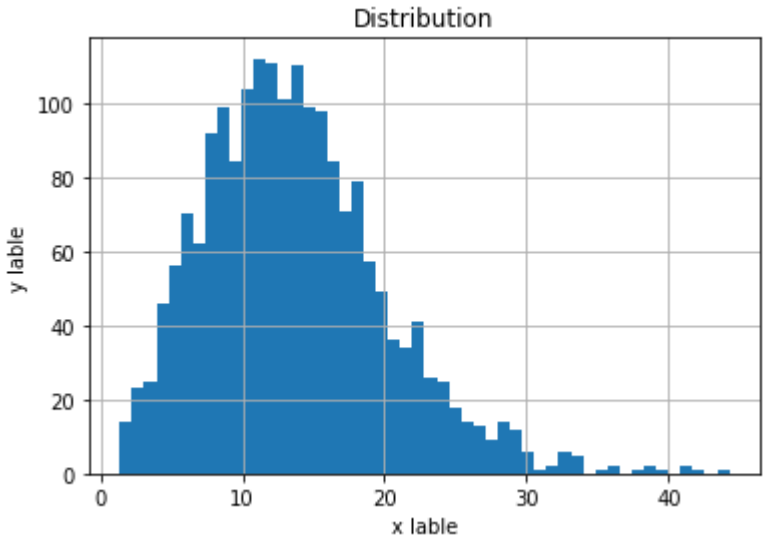
In [ ]:
```
# apply normalization technique, using a range of [0,1]
murders['PctEmplProfServ_norm'] = (murders['PctEmplProfServ'] - murders['PctEmplProfSe

murders.head()
```

Out[ ]:

	communityname	state	population	householdsize	racepctblack	racePctWhite	racePctAsian
0	BerkeleyHeightstownship	NJ	11980	3.10	1.37	91.78	6.50
1	Marpletownship	PA	23123	2.82	0.80	95.57	3.44
2	Tigardcity	OR	29344	2.43	0.74	94.33	3.43
4	Springfieldcity	MO	140494	2.45	2.51	95.65	0.90
5	Norwoodtown	MA	28700	2.60	1.60	96.57	1.47

In [ ]:
```
murders = murders.drop(['PctEmplProfServ_bins'], axis=1)
```

In [ ]:
```
murders['PctOccupManu'].hist(bins=50)
plt.title('Distribution')
plt.xlabel('x lable')
plt.ylabel('y lable')
plt.show()
```

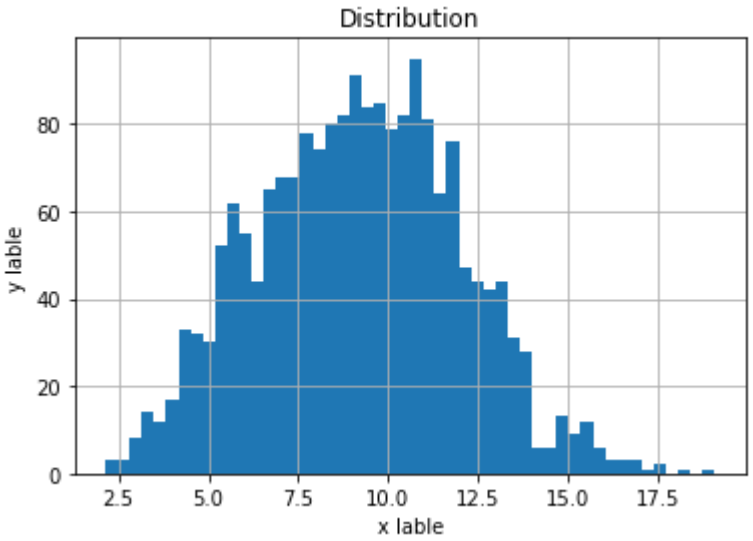


In [ ]:
```
# apply normalization technique, using a range of [0,1]
murders['PctOccupManu_norm'] = (murders['PctOccupManu'] - murders['PctOccupManu'].min(

murders.head()
```

Out[ ]:

	communityname	state	population	householdsize	racepctblack	racePctWhite	racePctAsian
0	BerkeleyHeightstownship	NJ	11980	3.10	1.37	91.78	6.50
1	Marpletownship	PA	23123	2.82	0.80	95.57	3.44
2	Tigardcity	OR	29344	2.43	0.74	94.33	3.43
4	Springfieldcity	MO	140494	2.45	2.51	95.65	0.90
5	Norwoodtown	MA	28700	2.60	1.60	96.57	1.47

In [ ]:
```
murders['MalePctDivorce'].hist(bins=50)
plt.title('Distribution')
plt.xlabel('x lable')
plt.ylabel('y lable')
plt.show()
```

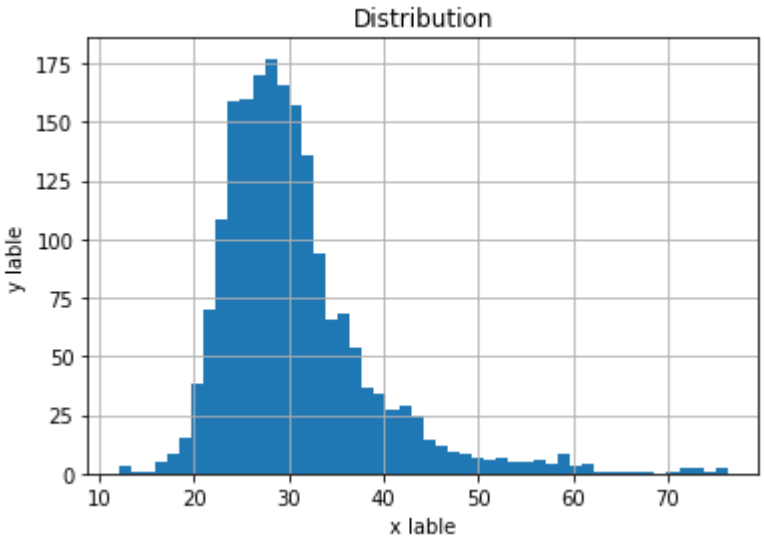


In [ ]:
```
# apply normalization technique, using a range of [0,1]
murders['MalePctDivorce_norm'] = (murders['MalePctDivorce'] - murders['MalePctDivorce'

murders.head()
```

Out[ ]:

	communityname	state	population	householdsize	racepctblack	racePctWhite	racePctAsian
0	BerkeleyHeightstownship	NJ	11980	3.10	1.37	91.78	6.50
1	Marpletownship	PA	23123	2.82	0.80	95.57	3.44
2	Tigardcity	OR	29344	2.43	0.74	94.33	3.43
4	Springfieldcity	MO	140494	2.45	2.51	95.65	0.90
5	Norwoodtown	MA	28700	2.60	1.60	96.57	1.47

In [ ]:
```
murders['MalePctNevMarr'].hist(bins=50)
plt.title('Distribution')
plt.xlabel('x lable')
plt.ylabel('y lable')
plt.show()
```



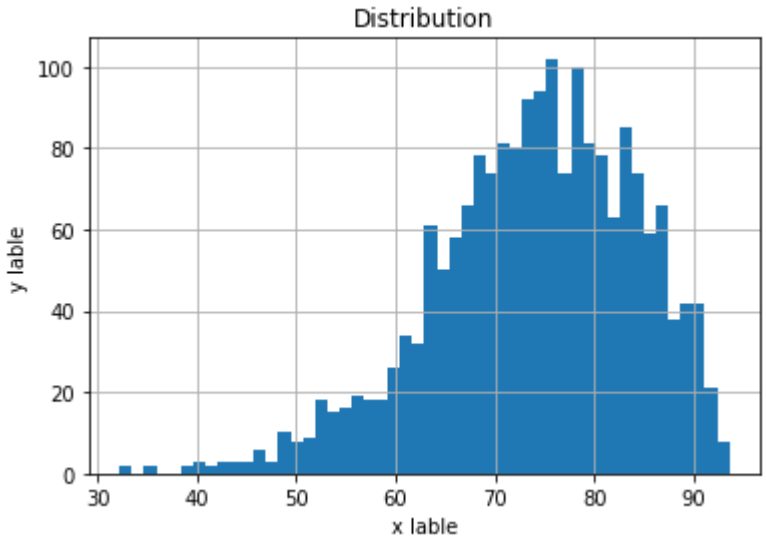
In [ ]:
```
# apply normalization technique, using a range of [0,1]
murders['MalePctNevMarr_norm'] = (murders['MalePctNevMarr'] - murders['MalePctNevMarr'

murders.head()
```

Out[ ]:

	communityname	state	population	householdsize	racepctblack	racePctWhite	racePctAsian
0	BerkeleyHeightstownship	NJ	11980	3.10	1.37	91.78	6.50
1	Marpletownship	PA	23123	2.82	0.80	95.57	3.44
2	Tigardcity	OR	29344	2.43	0.74	94.33	3.43
4	Springfieldcity	MO	140494	2.45	2.51	95.65	0.90
5	Norwoodtown	MA	28700	2.60	1.60	96.57	1.47

In [ ]:
```
murders = murders.drop(['MalePctNevMarr_bins'], axis=1)
```

In [ ]:
```
murders['PctFam2Par'].hist(bins=50)
plt.title('Distribution')
plt.xlabel('x lable')
plt.ylabel('y lable')
plt.show()
```



In [ ]:
```
#Check the minimum and maximum value
print(murders['PctFam2Par'].min())
print(murders['PctFam2Par'].max())

#Determine bins based on quantiles
PctFam2Par_ = pd.qcut(murders['PctFam2Par'], q=4)
#Check the value counts of each bin to ensure they are balanced
PctFam2Par_.value_counts()

#Create bin labels
PctFam2Par_labels = ['0-67%', '67-70%', '75-82%', '82-95%']
#Create bins
PctFam2Par_bin = [0, 67, 75, 82, 95]
#Add new category
```

```
murders['PctFam2Par_bins'] = pd.cut(murders['PctFam2Par'], bins=PctFam2Par_bin, labels
murders.head()
```

```
32.24
93.6
```

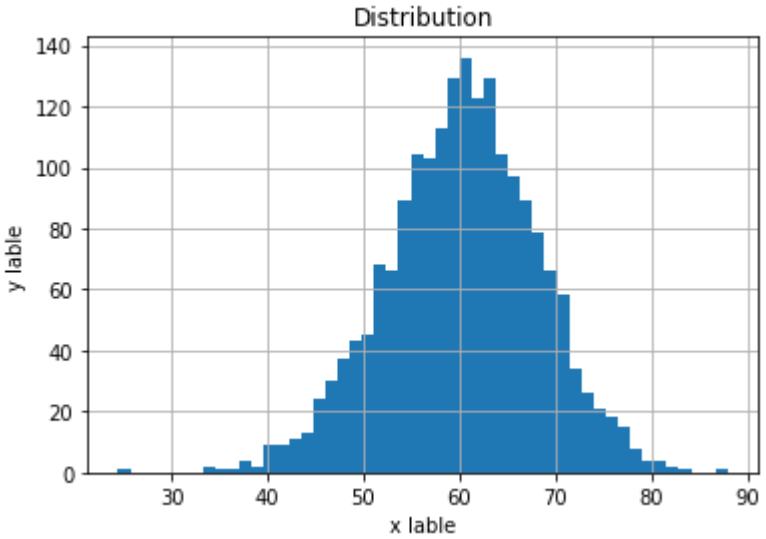
Out[ ]:

	communityname	state	population	householdsize	racepctblack	racePctWhite	racePctAsian
0	BerkeleyHeightstownship	NJ	11980	3.10	1.37	91.78	6.50
1	Marpletownship	PA	23123	2.82	0.80	95.57	3.44
2	Tigardcity	OR	29344	2.43	0.74	94.33	3.43
4	Springfieldcity	MO	140494	2.45	2.51	95.65	0.90
5	Norwoodtown	MA	28700	2.60	1.60	96.57	1.47

In [ ]:
```
murders['PctFam2Par_bins'].value_counts()
```

Out[ ]:
```
67-70%    533
75-82%    482
82-95%    462
0-67%     442
Name: PctFam2Par_bins, dtype: int64
```

In [ ]:
```
murders['PctWorkMomYoungKids'].hist(bins=50)
plt.title('Distribution')
plt.xlabel('x lable')
plt.ylabel('y lable')
plt.show()
```

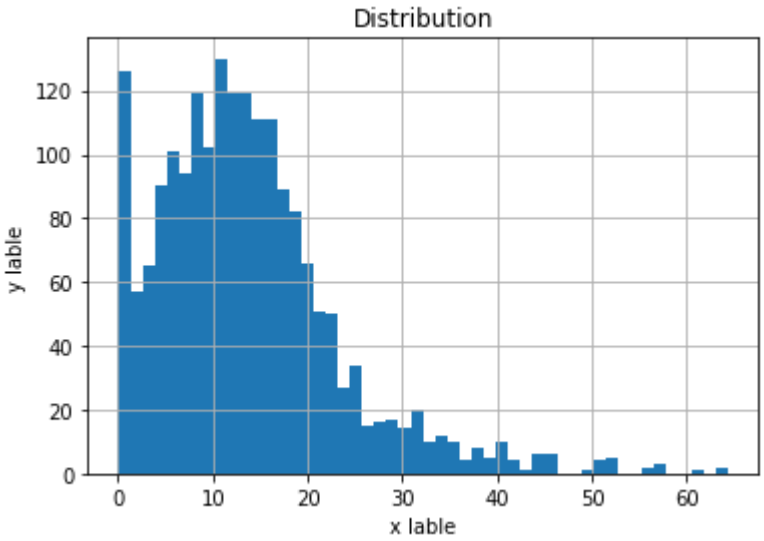


In [ ]:
```
# apply normalization technique, using a range of [0,1]
murders['PctWorkMomYoungKids_norm'] = (murders['PctWorkMomYoungKids'] - murders['PctWo

murders.head()
```

Out[ ]:

	communityname	state	population	householdsize	racepctblack	racePctWhite	racePctAsian
0	BerkeleyHeightstownship	NJ	11980	3.10	1.37	91.78	6.50
1	Marpletownship	PA	23123	2.82	0.80	95.57	3.44
2	Tigardcity	OR	29344	2.43	0.74	94.33	3.43
4	Springfieldcity	MO	140494	2.45	2.51	95.65	0.90
5	Norwoodtown	MA	28700	2.60	1.60	96.57	1.47

In [ ]:
```
murders['PctImmigRecent'].hist(bins=50)
plt.title('Distribution')
plt.xlabel('x lable')
plt.ylabel('y lable')
plt.show()
```



In [ ]:
```
#Check the minimum and maximum value
print(murders['PctImmigRecent'].min())
print(murders['PctImmigRecent'].max())

#Determine bins based on quantiles
PctImmigRecent_ = pd.qcut(murders['PctImmigRecent'], q=4)
#Check the value counts of each bin to ensure they are balanced
PctImmigRecent_.value_counts()

#Create bin labels
PctImmigRecent_labels = ['0-7%', '7-12%', '12-18%', '18-65%']
#Create bins
PctImmigRecent_bin = [-1, 7, 12, 18, 65]
#Add new category
murders['PctImmigRecent_bins'] = pd.cut(murders['PctImmigRecent'], bins=PctImmigRecent
murders.head()
```

```
0.0
64.29
```

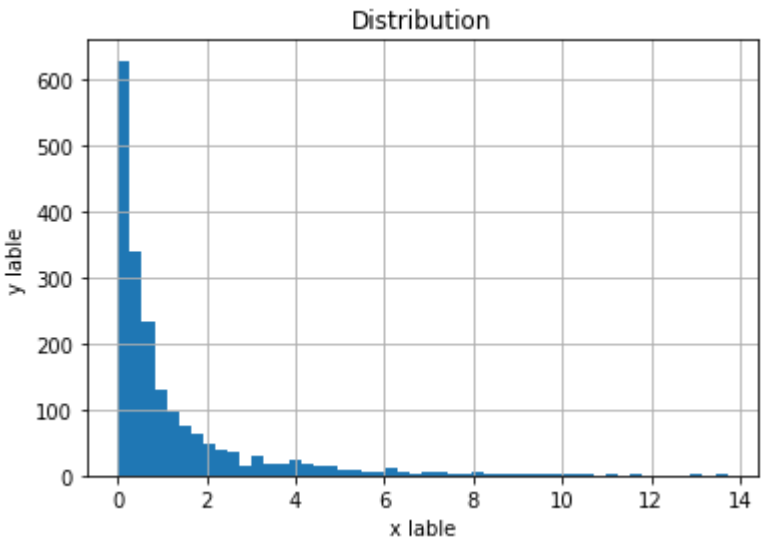
Out[ ]:

	communityname	state	population	householdsize	racepctblack	racePctWhite	racePctAsian
0	BerkeleyHeightstownship	NJ	11980	3.10	1.37	91.78	6.50
1	Marpletownship	PA	23123	2.82	0.80	95.57	3.44
2	Tigardcity	OR	29344	2.43	0.74	94.33	3.43
4	Springfieldcity	MO	140494	2.45	2.51	95.65	0.90
5	Norwoodtown	MA	28700	2.60	1.60	96.57	1.47

In [ ]:
```
murders['PctImmigRecent_bins'].value_counts()
```

Out[ ]:
```
12-18%    515
18-65%    486
0-7%      482
7-12%     436
Name: PctImmigRecent_bins, dtype: int64
```

In [ ]:
```
murders['PctRecentImmig'].hist(bins=50)
plt.title('Distribution')
plt.xlabel('x lable')
plt.ylabel('y lable')
plt.show()
```



In [ ]:
```
#Check the minimum and maximum value
print(murders['PctRecentImmig'].min())
print(murders['PctRecentImmig'].max())

#Determine bins based on quantiles
PctRecentImmig_ = pd.qcut(murders['PctRecentImmig'], q=4)
#Check the value counts of each bin to ensure they are balanced
```

```
PctRecentImmig_.value_counts()

#Create bin labels
PctRecentImmig_labels = ['0-0.2%', '0.2-0.5%', '0.5-1.3%', '1.3-13.7%']
#Create bins
PctRecentImmig_bin = [-1, 0.2, 0.5, 1.3, 13.71]
#Add new category
murders['PctRecentImmig_bins'] = pd.cut(murders['PctRecentImmig'], bins=PctRecentImmig
murders.head()
```

```
0.0
13.71
```

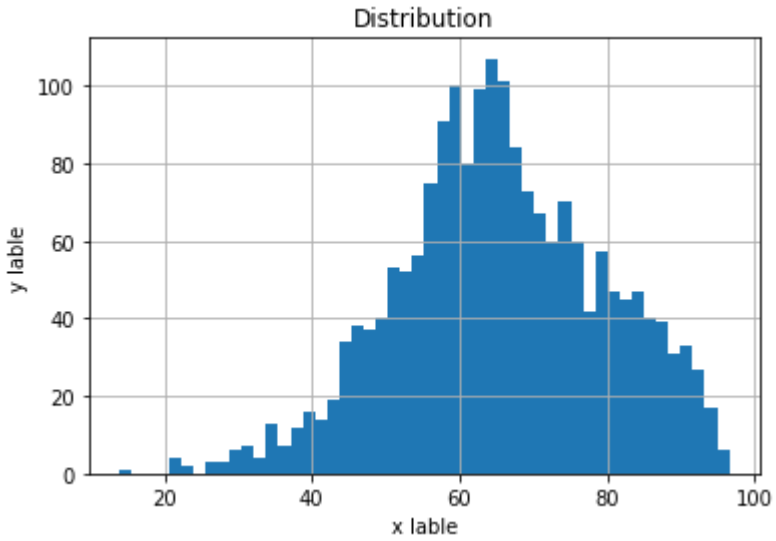
Out[ ]:

	communityname	state	population	householdsize	racepctblack	racePctWhite	racePctAsian
0	BerkeleyHeightstownship	NJ	11980	3.10	1.37	91.78	6.50
1	Marpletownship	PA	23123	2.82	0.80	95.57	3.44
2	Tigardcity	OR	29344	2.43	0.74	94.33	3.43
4	Springfieldcity	MO	140494	2.45	2.51	95.65	0.90
5	Norwoodtown	MA	28700	2.60	1.60	96.57	1.47

In [ ]:
```
murders['PctRecentImmig_bins'].value_counts()
```

Out[ ]:
```
0-0.2%       526
1.3-13.7%    516
0.5-1.3%     474
0.2-0.5%     403
Name: PctRecentImmig_bins, dtype: int64
```

In [ ]:
```
murders['PctPersOwnOccup'].hist(bins=50)
plt.title('Distribution')
plt.xlabel('x lable')
plt.ylabel('y lable')
plt.show()
```

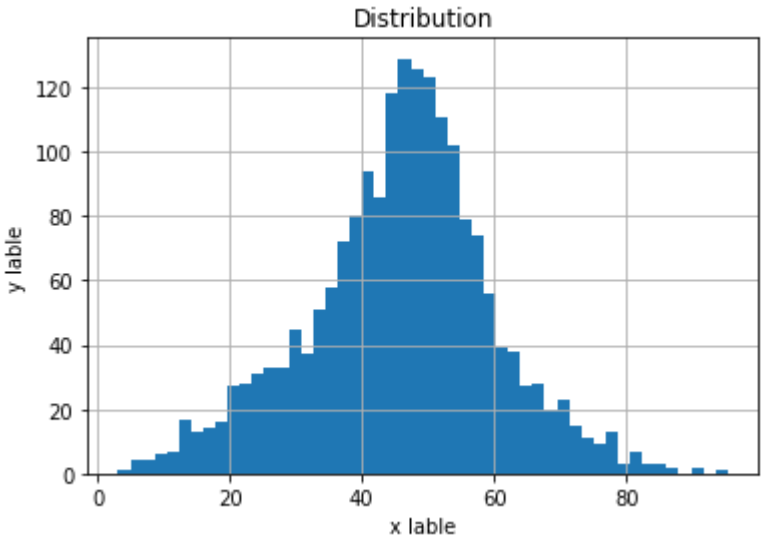


In [ ]:
```
# apply normalization technique, using a range of [0,1]
murders['PctPersOwnOccup_norm'] = (murders['PctPersOwnOccup'] - murders['PctPersOwnOcc

murders.head()
```

Out[ ]:

	communityname	state	population	householdsize	racepctblack	racePctWhite	racePctAsian
0	BerkeleyHeightstownship	NJ	11980	3.10	1.37	91.78	6.50
1	Marpletownship	PA	23123	2.82	0.80	95.57	3.44
2	Tigardcity	OR	29344	2.43	0.74	94.33	3.43
4	Springfieldcity	MO	140494	2.45	2.51	95.65	0.90
5	Norwoodtown	MA	28700	2.60	1.60	96.57	1.47

In [ ]:
```
murders['PctHousLess3BR'].hist(bins=50)
plt.title('Distribution')
plt.xlabel('x lable')
plt.ylabel('y lable')
plt.show()
```

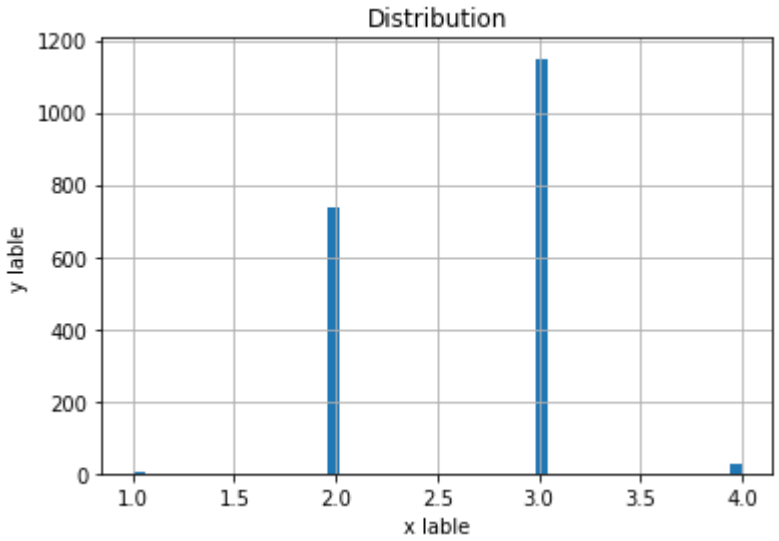


In [ ]:
```
# apply normalization technique, using a range of [0,1]
murders['PctHousLess3BR_norm'] = (murders['PctHousLess3BR'] - murders['PctHousLess3BR'

murders.head()
```

Out[ ]:

	communityname	state	population	householdsize	racepctblack	racePctWhite	racePctAsian
0	BerkeleyHeightstownship	NJ	11980	3.10	1.37	91.78	6.50
1	Marpletownship	PA	23123	2.82	0.80	95.57	3.44
2	Tigardcity	OR	29344	2.43	0.74	94.33	3.43
4	Springfieldcity	MO	140494	2.45	2.51	95.65	0.90
5	Norwoodtown	MA	28700	2.60	1.60	96.57	1.47

In [ ]:
```
murders['MedNumBR'].hist(bins=50)
plt.title('Distribution')
plt.xlabel('x lable')
plt.ylabel('y lable')
plt.show()
```



In [ ]:
```
#Check the minimum and maximum value
print(murders['MedNumBR'].min())
print(murders['MedNumBR'].max())

#Determine bins based on quantiles
murders['MedNumBR'].describe()

#Create bin labels
MedNumBR_labels = ['0-2.5', '2.5-4']
#Create bins
MedNumBR_bin = [0, 2.5, 4]
#Add new category
murders['MedNumBR_bins'] = pd.cut(murders['MedNumBR'], bins=MedNumBR_bin, labels=MedNu
murders.head()
```

```
1
4
```

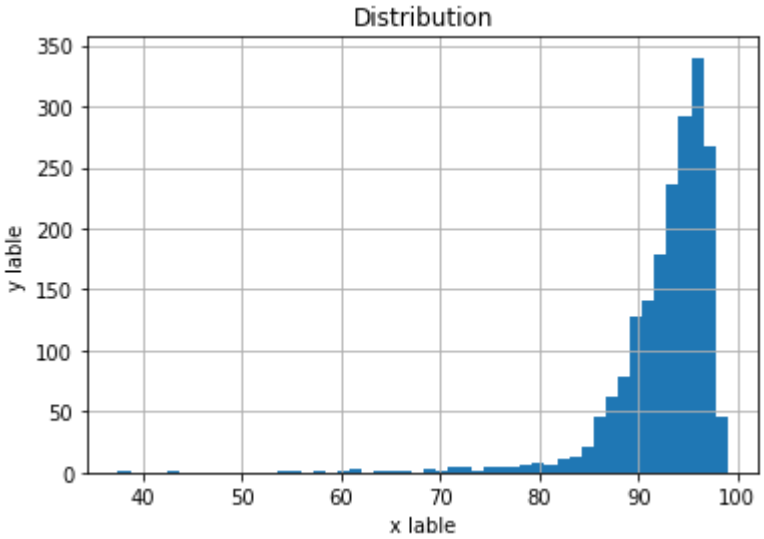
Out[ ]:

	communityname	state	population	householdsize	racepctblack	racePctWhite	racePctAsian
0	BerkeleyHeightstownship	NJ	11980	3.10	1.37	91.78	6.50
1	Marpletownship	PA	23123	2.82	0.80	95.57	3.44
2	Tigardcity	OR	29344	2.43	0.74	94.33	3.43
4	Springfieldcity	MO	140494	2.45	2.51	95.65	0.90
5	Norwoodtown	MA	28700	2.60	1.60	96.57	1.47

In [ ]:
```
murders['MedNumBR_bins'].value_counts()
```

Out[ ]:
```
2.5-4    1176
0-2.5     743
Name: MedNumBR_bins, dtype: int64
```

In [ ]:
```
murders['PctHousOccup'].hist(bins=50)
plt.title('Distribution')
plt.xlabel('x lable')
plt.ylabel('y lable')
plt.show()
```



In [ ]:
```
#Check the minimum and maximum value
print(murders['PctHousOccup'].min())
print(murders['PctHousOccup'].max())

#Determine bins based on quantiles
PctHousOccup_ = pd.qcut(murders['PctHousOccup'], q=4)
#Check the value counts of each bin to ensure they are balanced
PctHousOccup_.value_counts()

#Create bin labels
PctHousOccup_labels = ['30-91%', '91-94%', '94-96%', '96-99%']
#Create bins
PctHousOccup_bin = [30, 91, 94, 96, 99]
#Add new category
murders['PctHousOccup_bins'] = pd.cut(murders['PctHousOccup'], bins=PctHousOccup_bin,
murders.head()
```

```
37.47
99.0
```

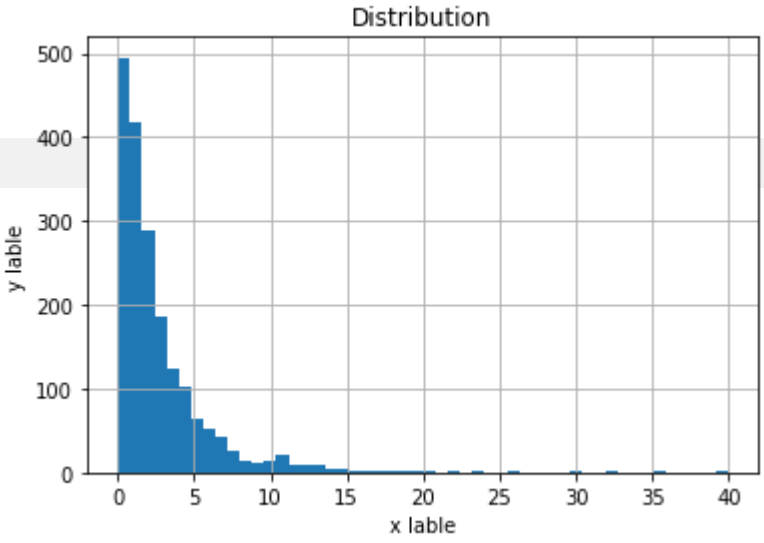
Out[ ]:

	communityname	state	population	householdsize	racepctblack	racePctWhite	racePctAsian
0	BerkeleyHeightstownship	NJ	11980	3.10	1.37	91.78	6.50
1	Marpletownship	PA	23123	2.82	0.80	95.57	3.44
2	Tigardcity	OR	29344	2.43	0.74	94.33	3.43
4	Springfieldcity	MO	140494	2.45	2.51	95.65	0.90
5	Norwoodtown	MA	28700	2.60	1.60	96.57	1.47

In [ ]:
```
murders['PctHousOccup_bins'].value_counts()
```

Out[ ]:
```
94-96%    495
30-91%    486
91-94%    475
96-99%    463
Name: PctHousOccup_bins, dtype: int64
```

In [ ]:
```
murders['PctVacantBoarded'].hist(bins=50)
plt.title('Distribution')
plt.xlabel('x lable')
plt.ylabel('y lable')
plt.show()
```



In [ ]:
```
#Check the minimum and maximum value
print(murders['PctVacantBoarded'].min())
print(murders['PctVacantBoarded'].max())

#Determine bins based on quantiles
PctVacantBoarded_ = pd.qcut(murders['PctVacantBoarded'], q=4)
#Check the value counts of each bin to ensure they are balanced
PctVacantBoarded_.value_counts()
```

```
#Create bin labels
PctVacantBoarded_labels = ['0-0.75%', '0.75-1.75%', '1.75-4%', '4-40%']
#Create bins
PctVacantBoarded_bin = [-1, 0.75, 1.75, 4, 40]
#Add new category
murders['PctVacantBoarded_bins'] = pd.cut(murders['PctVacantBoarded'], bins=PctVacantE
murders.head()
```

```
0.0
39.89
```

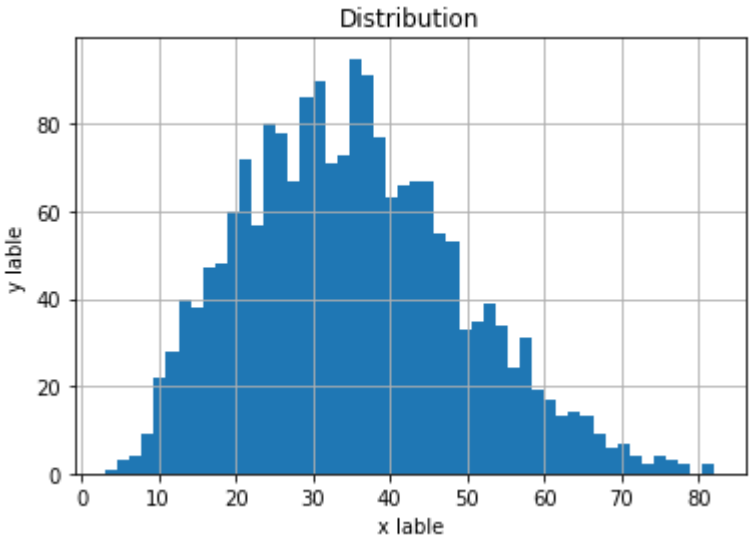
Out[ ]:

	communityname	state	population	householdsize	racepctblack	racePctWhite	racePctAsian
0	BerkeleyHeightstownship	NJ	11980	3.10	1.37	91.78	6.50
1	Marpletownship	PA	23123	2.82	0.80	95.57	3.44
2	Tigardcity	OR	29344	2.43	0.74	94.33	3.43
4	Springfieldcity	MO	140494	2.45	2.51	95.65	0.90
5	Norwoodtown	MA	28700	2.60	1.60	96.57	1.47

In [ ]:
```
murders['PctVacantBoarded_bins'].value_counts()
```

Out[ ]:
```
1.75-4%       540
0.75-1.75%    500
0-0.75%       473
4-40%         406
Name: PctVacantBoarded_bins, dtype: int64
```

In [ ]:
```
murders['PctVacMore6Mos'].hist(bins=50)
plt.title('Distribution')
plt.xlabel('x lable')
plt.ylabel('y lable')
plt.show()
```

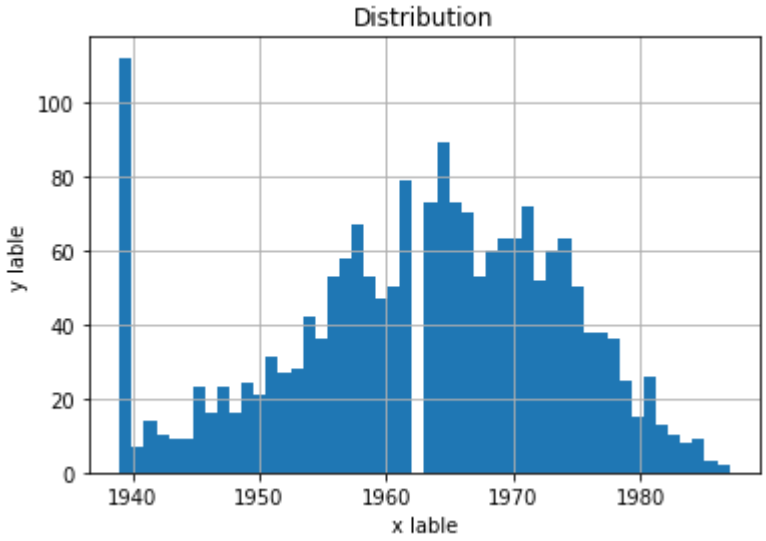


In [ ]:
```
# apply normalization technique, using a range of [0,1]
murders['PctVacMore6Mos_norm'] = (murders['PctVacMore6Mos'] - murders['PctVacMore6Mos'

murders.head()
```

Out[ ]:

	communityname	state	population	householdsize	racepctblack	racePctWhite	racePctAsian
0	BerkeleyHeightstownship	NJ	11980	3.10	1.37	91.78	6.50
1	Marpletownship	PA	23123	2.82	0.80	95.57	3.44
2	Tigardcity	OR	29344	2.43	0.74	94.33	3.43
4	Springfieldcity	MO	140494	2.45	2.51	95.65	0.90
5	Norwoodtown	MA	28700	2.60	1.60	96.57	1.47

In [ ]:
```
murders['MedYrHousBuilt'].hist(bins=50)
plt.title('Distribution')
plt.xlabel('x lable')
plt.ylabel('y lable')
plt.show()
```



In [ ]:
```
#Check the minimum and maximum value
print(murders['MedYrHousBuilt'].min())
print(murders['MedYrHousBuilt'].max())

#Determine bins based on quantiles
MedYrHousBuilt_ = pd.qcut(murders['MedYrHousBuilt'], q=4)
#Check the value counts of each bin to ensure they are balanced
MedYrHousBuilt_.value_counts()

#Create bin labels
MedYrHousBuilt_labels = ['1939-1956', '1956-1964', '1964-1971', '1971-1987']
#Create bins
```

```
MedYrHousBuilt_bin = [0, 1956, 1964, 1971, 1987]
#Add new category
murders['MedYrHousBuilt_bins'] = pd.cut(murders['MedYrHousBuilt'], bins=MedYrHousBuilt
murders.head()
```

```
1939
1987
```

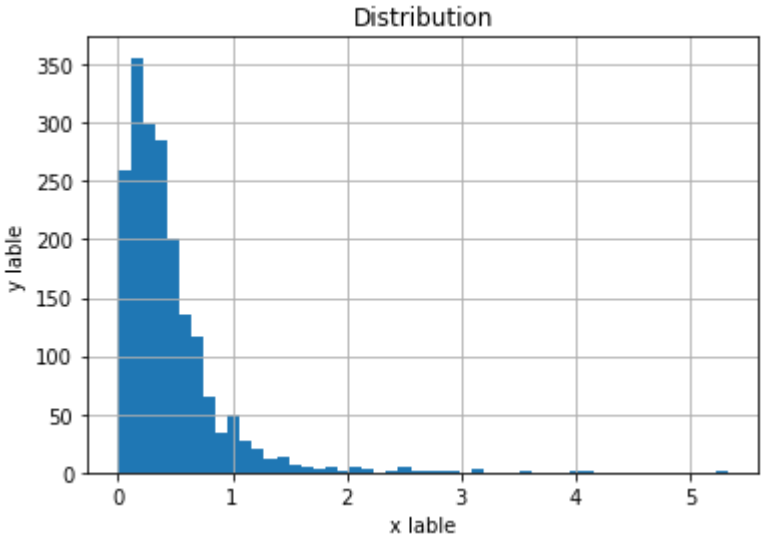
Out[ ]:

	communityname	state	population	householdsize	racepctblack	racePctWhite	racePctAsian
0	BerkeleyHeightstownship	NJ	11980	3.10	1.37	91.78	6.50
1	Marpletownship	PA	23123	2.82	0.80	95.57	3.44
2	Tigardcity	OR	29344	2.43	0.74	94.33	3.43
4	Springfieldcity	MO	140494	2.45	2.51	95.65	0.90
5	Norwoodtown	MA	28700	2.60	1.60	96.57	1.47

In [ ]:
```
murders['MedYrHousBuilt_bins'].value_counts()
```

Out[ ]:
```
1956-1964    516
1939-1956    501
1964-1971    454
1971-1987    448
Name: MedYrHousBuilt_bins, dtype: int64
```

In [ ]:
```
murders['PctWOFullPlumb'].hist(bins=50)
plt.title('Distribution')
plt.xlabel('x lable')
plt.ylabel('y lable')
plt.show()
```



In [ ]:
```
#Check the minimum and maximum value
print(murders['PctWOFullPlumb'].min())
print(murders['PctWOFullPlumb'].max())
```

```
#Determine bins based on quantiles
PctWOFullPlumb_ = pd.qcut(murders['PctWOFullPlumb'], q=4)
#Check the value counts of each bin to ensure they are balanced
PctWOFullPlumb_.value_counts()

#Create bin labels
PctWOFullPlumb_labels = ['0-0.16%', '0.16-0.3%', '0.3-0.5%', '0.5-5%']
#Create bins
PctWOFullPlumb_bin = [-1, 0.16, 0.3, 0.5, 5.33]
#Add new category
murders['PctWOFullPlumb_bins'] = pd.cut(murders['PctWOFullPlumb'], bins=PctWOFullPlumb
murders.head()
```

```
0.0
5.33
```

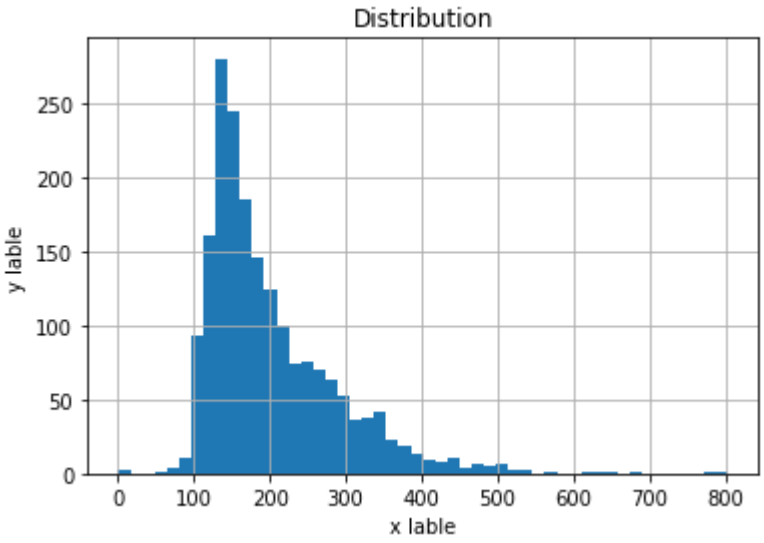
Out[ ]:

	communityname	state	population	householdsize	racepctblack	racePctWhite	racePctAsian
0	BerkeleyHeightstownship	NJ	11980	3.10	1.37	91.78	6.50
1	Marpletownship	PA	23123	2.82	0.80	95.57	3.44
2	Tigardcity	OR	29344	2.43	0.74	94.33	3.43
4	Springfieldcity	MO	140494	2.45	2.51	95.65	0.90
5	Norwoodtown	MA	28700	2.60	1.60	96.57	1.47

In [ ]:
```
murders['PctWOFullPlumb_bins'].value_counts()
```

Out[ ]:
```
0.5-5%       564
0.3-0.5%     475
0-0.16%      453
0.16-0.3%    427
Name: PctWOFullPlumb_bins, dtype: int64
```

In [ ]:
```
murders['RentQrange'].hist(bins=50)
plt.title('Distribution')
plt.xlabel('x lable')
plt.ylabel('y lable')
plt.show()
```



In [ ]:
```
#Check the minimum and maximum value
print(murders['RentQrange'].min())
print(murders['RentQrange'].max())

#Determine bins based on quantiles
RentQrange_ = pd.qcut(murders['RentQrange'], q=4)
#Check the value counts of each bin to ensure they are balanced
RentQrange_.value_counts()


#Create bin labels
RentQrange_labels = ['0-140', '140-170', '170-230', '230-805']
#Create bins
RentQrange_bin = [-1, 140, 170, 230, 803]
#Add new category
murders['RentQrange_bins'] = pd.cut(murders['RentQrange'], bins=RentQrange_bin, labels
murders.head()
```

```
0
803
```

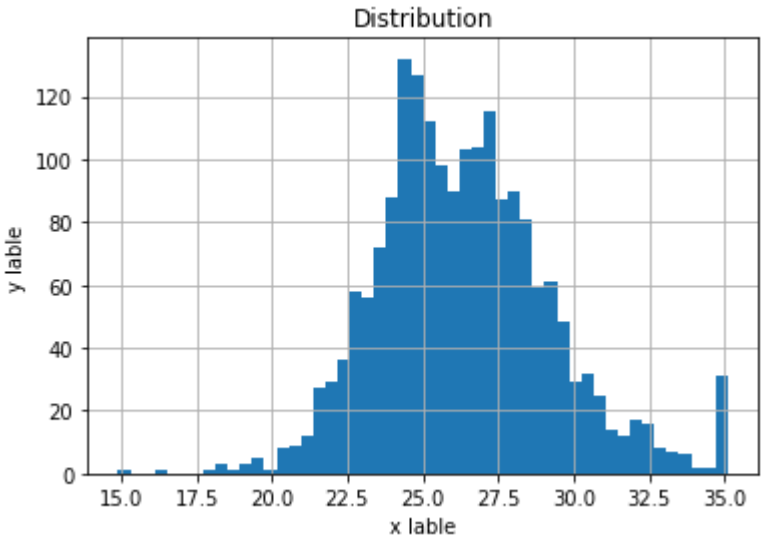
Out[ ]:

	communityname	state	population	householdsize	racepctblack	racePctWhite	racePctAsian
0	BerkeleyHeightstownship	NJ	11980	3.10	1.37	91.78	6.50
1	Marpletownship	PA	23123	2.82	0.80	95.57	3.44
2	Tigardcity	OR	29344	2.43	0.74	94.33	3.43
4	Springfieldcity	MO	140494	2.45	2.51	95.65	0.90
5	Norwoodtown	MA	28700	2.60	1.60	96.57	1.47

In [ ]:
```
murders['RentQrange_bins'].value_counts()
```

Out[ ]:
```
230-805    534
0-140      489
170-230    482
140-170    414
Name: RentQrange_bins, dtype: int64
```

In [ ]:
```
murders['MedRentPctHousInc'].hist(bins=50)
plt.title('Distribution')
plt.xlabel('x lable')
plt.ylabel('y lable')
plt.show()
```

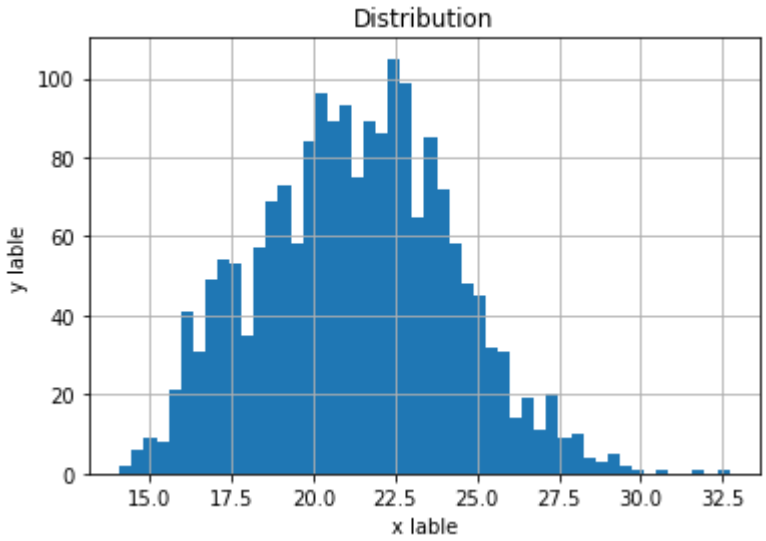


In [ ]:
```
# apply normalization technique, using a range of [0,1]
murders['MedRentPctHousInc_norm'] = (murders['MedRentPctHousInc'] - murders['MedRentPc

murders.head()
```

Out[ ]:

	communityname	state	population	householdsize	racepctblack	racePctWhite	racePctAsian
0	BerkeleyHeightstownship	NJ	11980	3.10	1.37	91.78	6.50
1	Marpletownship	PA	23123	2.82	0.80	95.57	3.44
2	Tigardcity	OR	29344	2.43	0.74	94.33	3.43
4	Springfieldcity	MO	140494	2.45	2.51	95.65	0.90
5	Norwoodtown	MA	28700	2.60	1.60	96.57	1.47

In [ ]:
```
murders['MedOwnCostPctInc'].hist(bins=50)
plt.title('Distribution')
plt.xlabel('x lable')
plt.ylabel('y lable')
plt.show()
```

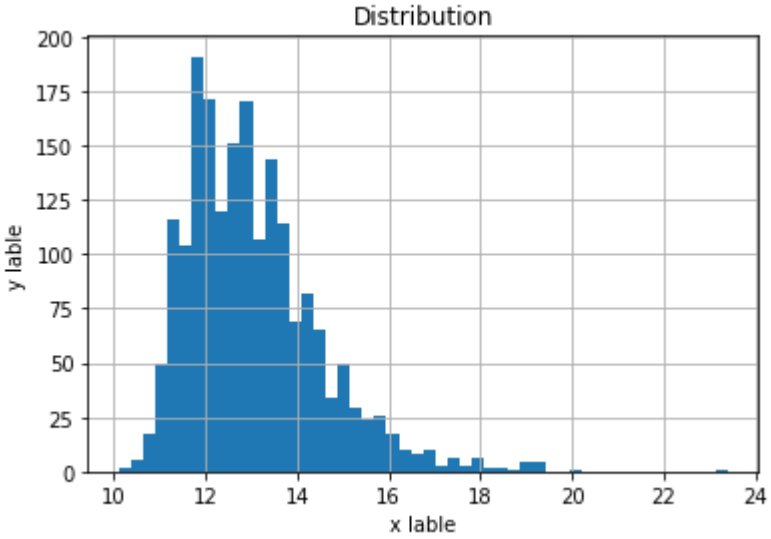


In [ ]:
```
# apply normalization technique, using a range of [0,1]
murders['MedOwnCostPctInc_norm'] = (murders['MedOwnCostPctInc'] - murders['MedOwnCostP

murders.head()
```

Out[ ]:

	communityname	state	population	householdsize	racepctblack	racePctWhite	racePctAsian
0	BerkeleyHeightstownship	NJ	11980	3.10	1.37	91.78	6.50
1	Marpletownship	PA	23123	2.82	0.80	95.57	3.44
2	Tigardcity	OR	29344	2.43	0.74	94.33	3.43
4	Springfieldcity	MO	140494	2.45	2.51	95.65	0.90
5	Norwoodtown	MA	28700	2.60	1.60	96.57	1.47

In [ ]:
```
murders['MedOwnCostPctIncNoMtg'].hist(bins=50)
plt.title('Distribution')
plt.xlabel('x lable')
plt.ylabel('y lable')
plt.show()
```



In [ ]:
```
#Check the minimum and maximum value
print(murders['MedOwnCostPctIncNoMtg'].min())
print(murders['MedOwnCostPctIncNoMtg'].max())

#Determine bins based on quantiles
MedOwnCostPctIncNoMtg_ = pd.qcut(murders['MedOwnCostPctIncNoMtg'], q=4)
#Check the value counts of each bin to ensure they are balanced
MedOwnCostPctIncNoMtg_.value_counts()

#Create bin labels
MedOwnCostPctIncNoMtg_labels = ['10-12%', '12-13%', '13-14%', '14-25%']
#Create bins
MedOwnCostPctIncNoMtg_bin = [10, 12, 13, 14, 25]
#Add new category
murders['MedOwnCostPctIncNoMtg_bins'] = pd.cut(murders['MedOwnCostPctIncNoMtg'], bins=
murders.head()
```

```
10.1
23.4
```

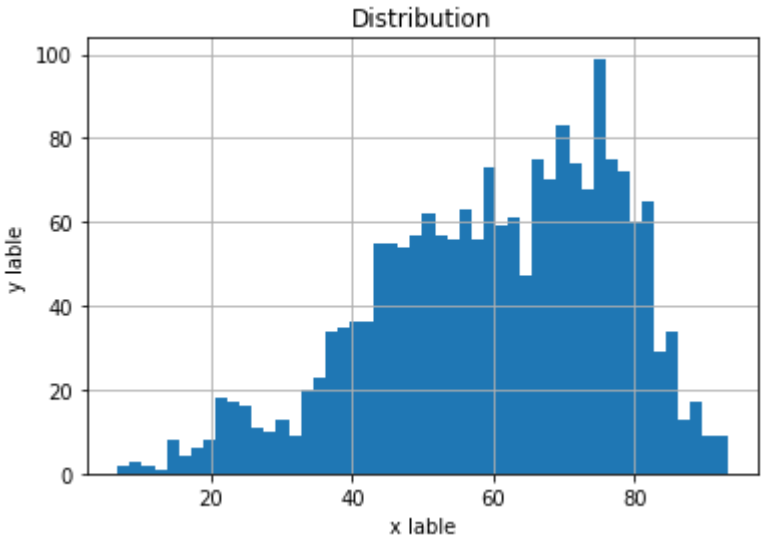
Out[ ]:

	communityname	state	population	householdsize	racepctblack	racePctWhite	racePctAsian
0	BerkeleyHeightstownship	NJ	11980	3.10	1.37	91.78	6.50
1	Marpletownship	PA	23123	2.82	0.80	95.57	3.44
2	Tigardcity	OR	29344	2.43	0.74	94.33	3.43
4	Springfieldcity	MO	140494	2.45	2.51	95.65	0.90
5	Norwoodtown	MA	28700	2.60	1.60	96.57	1.47

In [ ]:
```
murders['MedOwnCostPctIncNoMtg_bins'].value_counts()
```

Out[ ]:
```
12-13%    559
10-12%    538
13-14%    434
14-25%    388
Name: MedOwnCostPctIncNoMtg_bins, dtype: int64
```

In [ ]:
```
murders['PctBornSameState'].hist(bins=50)
plt.title('Distribution')
plt.xlabel('x lable')
plt.ylabel('y lable')
plt.show()
```



In [ ]:
```
#Check the minimum and maximum value
print(murders['PctBornSameState'].min())
print(murders['PctBornSameState'].max())

#Determine bins based on quantiles
PctBornSameState_ = pd.qcut(murders['PctBornSameState'], q=4)
#Check the value counts of each bin to ensure they are balanced
PctBornSameState_.value_counts()

#Create bin labels
PctBornSameState_labels = ['0-50%', '50-65%', '65-75%', '75-95%']
#Create bins
PctBornSameState_bin = [0, 50, 65, 75, 95]
#Add new category
murders['PctBornSameState_bins'] = pd.cut(murders['PctBornSameState'], bins=PctBornSam
murders.head()
```

```
6.75
93.14
```

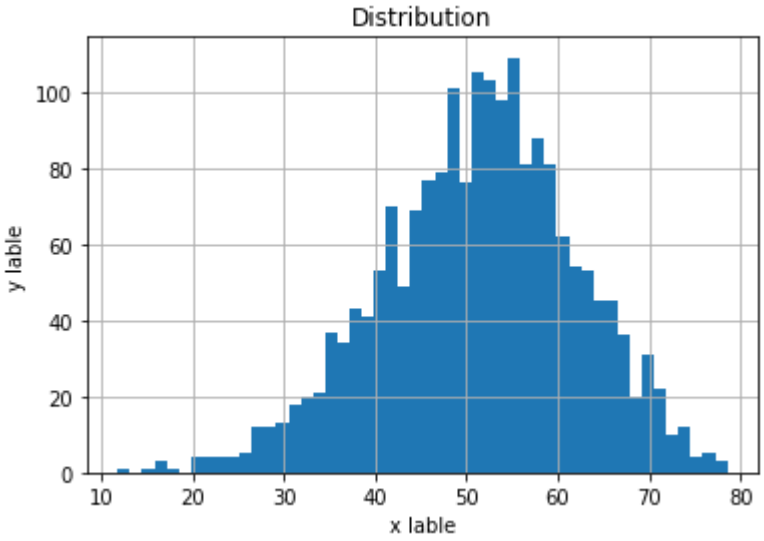
Out[ ]:

	communityname	state	population	householdsize	racepctblack	racePctWhite	racePctAsian
0	BerkeleyHeightstownship	NJ	11980	3.10	1.37	91.78	6.50
1	Marpletownship	PA	23123	2.82	0.80	95.57	3.44
2	Tigardcity	OR	29344	2.43	0.74	94.33	3.43
4	Springfieldcity	MO	140494	2.45	2.51	95.65	0.90
5	Norwoodtown	MA	28700	2.60	1.60	96.57	1.47

In [ ]:
```
murders['PctBornSameState_bins'].value_counts()
```

Out[ ]:
```
0-50%     535
50-65%    520
65-75%    433
75-95%    431
Name: PctBornSameState_bins, dtype: int64
```

In [ ]:
```
murders['PctSameHouse85'].hist(bins=50)
plt.title('Distribution')
plt.xlabel('x lable')
plt.ylabel('y lable')
plt.show()
```

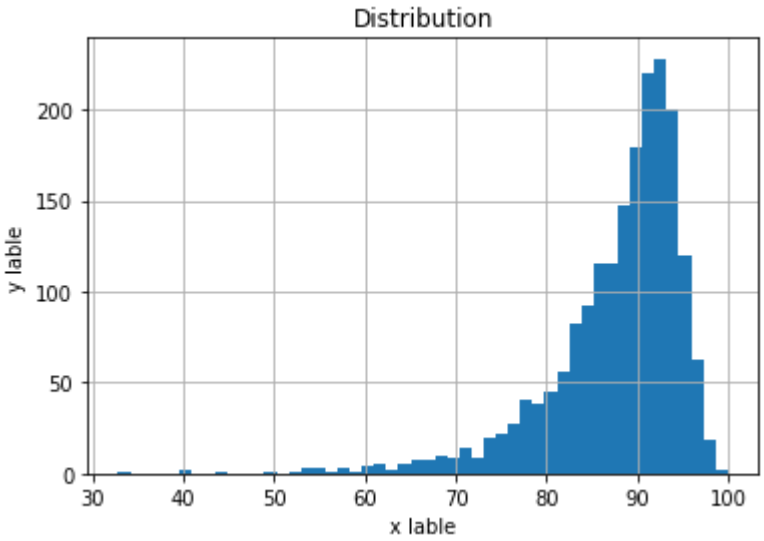


In [ ]:
```
# apply normalization technique, using a range of [0,1]
murders['PctSameHouse85_norm'] = (murders['PctSameHouse85'] - murders['PctSameHouse85'

murders.head()
```

Out[ ]:

	communityname	state	population	householdsize	racepctblack	racePctWhite	racePctAsian
0	BerkeleyHeightstownship	NJ	11980	3.10	1.37	91.78	6.50
1	Marpletownship	PA	23123	2.82	0.80	95.57	3.44
2	Tigardcity	OR	29344	2.43	0.74	94.33	3.43
4	Springfieldcity	MO	140494	2.45	2.51	95.65	0.90
5	Norwoodtown	MA	28700	2.60	1.60	96.57	1.47

In [ ]:
```
murders['PctSameState85'].hist(bins=50)
plt.title('Distribution')
plt.xlabel('x lable')
plt.ylabel('y lable')
plt.show()
```



In [ ]:
```
#Check the minimum and maximum value
print(murders['PctSameState85'].min())
print(murders['PctSameState85'].max())

#Determine bins based on quantiles
PctSameState85_ = pd.qcut(murders['PctSameState85'], q=4)
#Check the value counts of each bin to ensure they are balanced
PctSameState85_.value_counts()

#Create bin labels
PctSameState85_labels = ['32-85%', '85-90%', '90-93%', '93-100%']
#Create bins
PctSameState85_bin = [32, 85, 90, 93, 100]
#Add new category
murders['PctSameState85_bins'] = pd.cut(murders['PctSameState85'], bins=PctSameState85
murders.head()
```

```
32.83
99.9
```

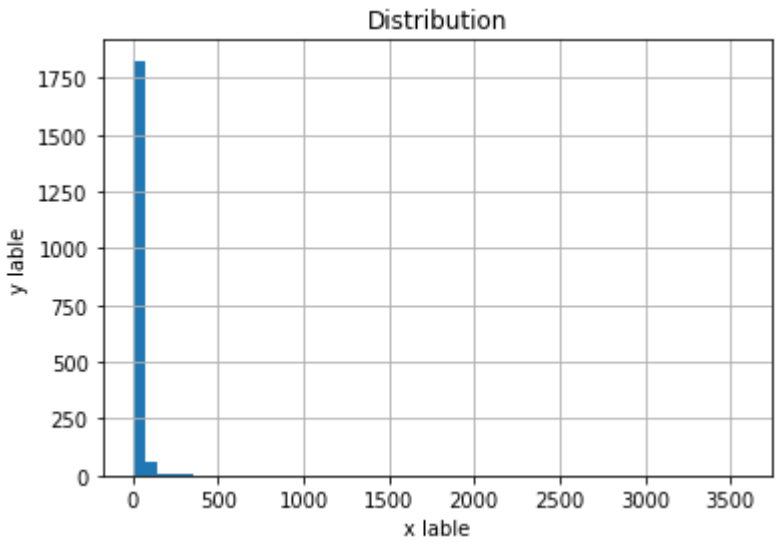
Out[ ]:

	communityname	state	population	householdsize	racepctblack	racePctWhite	racePctAsian
0	BerkeleyHeightstownship	NJ	11980	3.10	1.37	91.78	6.50
1	Marpletownship	PA	23123	2.82	0.80	95.57	3.44
2	Tigardcity	OR	29344	2.43	0.74	94.33	3.43
4	Springfieldcity	MO	140494	2.45	2.51	95.65	0.90
5	Norwoodtown	MA	28700	2.60	1.60	96.57	1.47

In [ ]:
```
murders['PctSameState85_bins'].value_counts()
```

Out[ ]:
```
32-85%     505
85-90%     500
90-93%     473
93-100%    441
Name: PctSameState85_bins, dtype: int64
```

In [ ]:
```
murders['LandArea'].hist(bins=50)
plt.title('Distribution')
plt.xlabel('x lable')
plt.ylabel('y lable')
plt.show()
```



In [ ]:
```
#Check the minimum and maximum value
print(murders['LandArea'].min())
print(murders['LandArea'].max())

#Determine bins based on quantiles
LandArea_ = pd.qcut(murders['LandArea'], q=4)
#Check the value counts of each bin to ensure they are balanced
```

```
LandArea_.value_counts()

#Create bin labels
LandArea_labels = ['0-7', '7-14', '14-26', '26-4000']
#Create bins
LandArea_bin = [0, 7, 14, 26, 3570]
#Add new category
murders['LandArea_bins'] = pd.cut(murders['LandArea'], bins=LandArea_bin, labels=LandA
murders.head()
```

```
0.9
3569.8
```

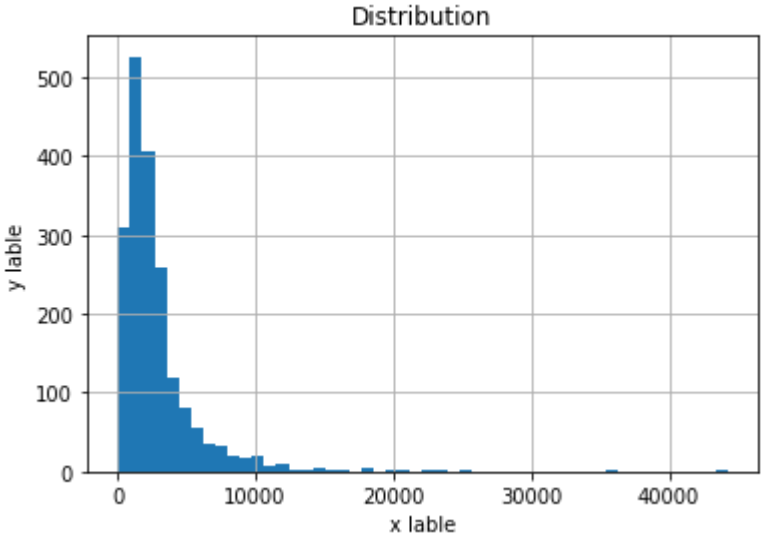
Out[ ]:

	communityname	state	population	householdsize	racepctblack	racePctWhite	racePctAsian
0	BerkeleyHeightstownship	NJ	11980	3.10	1.37	91.78	6.50
1	Marpletownship	PA	23123	2.82	0.80	95.57	3.44
2	Tigardcity	OR	29344	2.43	0.74	94.33	3.43
4	Springfieldcity	MO	140494	2.45	2.51	95.65	0.90
5	Norwoodtown	MA	28700	2.60	1.60	96.57	1.47

In [ ]:
```
murders['LandArea_bins'].value_counts()
```

Out[ ]:
```
7-14       525
0-7        467
26-4000    465
14-26      462
Name: LandArea_bins, dtype: int64
```

In [ ]:
```
murders['PopDens'].hist(bins=50)
plt.title('Distribution')
plt.xlabel('x lable')
plt.ylabel('y lable')
plt.show()
```



In [ ]:
```
#Check the minimum and maximum value
print(murders['PopDens'].min())
print(murders['PopDens'].max())

#Determine bins based on quantiles
PopDens_ = pd.qcut(murders['PopDens'], q=4)
#Check the value counts of each bin to ensure they are balanced
PopDens_.value_counts()

#Create bin labels
PopDens_labels = ['10-1200', '1200-2000', '2000-3300', '3300-45000']
#Create bins
PopDens_bin = [9, 1200, 2000, 3300, 45000]
#Add new category
murders['PopDens_bins'] = pd.cut(murders['PopDens'], bins=PopDens_bin, labels=PopDens_
murders.head()
```

```
10.0
44229.9
```

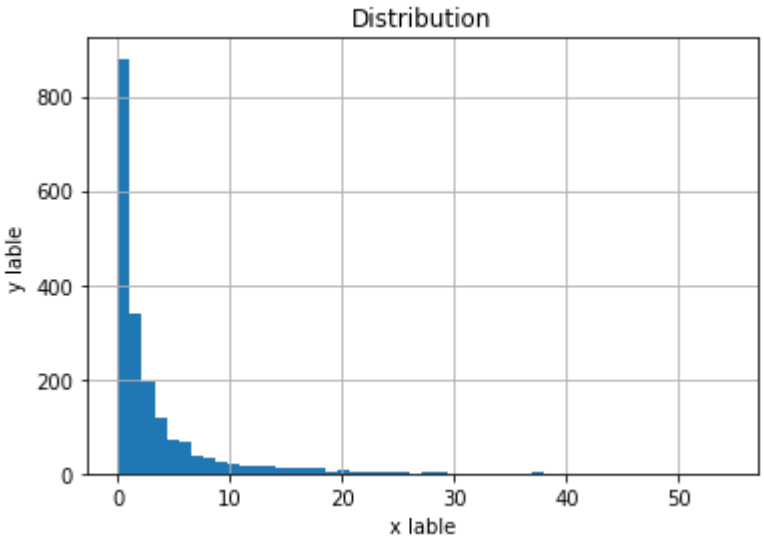
Out[ ]:

	communityname	state	population	householdsize	racepctblack	racePctWhite	racePctAsian
0	BerkeleyHeightstownship	NJ	11980	3.10	1.37	91.78	6.50
1	Marpletownship	PA	23123	2.82	0.80	95.57	3.44
2	Tigardcity	OR	29344	2.43	0.74	94.33	3.43
4	Springfieldcity	MO	140494	2.45	2.51	95.65	0.90
5	Norwoodtown	MA	28700	2.60	1.60	96.57	1.47

In [ ]:
```
murders['PopDens_bins'].value_counts()
```

Out[ ]:
```
10-1200       496
2000-3300     488
3300-45000    478
1200-2000     457
Name: PopDens_bins, dtype: int64
```

In [ ]:
```
murders['PctUsePubTrans'].hist(bins=50)
plt.title('Distribution')
plt.xlabel('x lable')
plt.ylabel('y lable')
plt.show()
```



In [ ]:
```
#Check the minimum and maximum value
print(murders['PctUsePubTrans'].min())
print(murders['PctUsePubTrans'].max())

#Determine bins based on quantiles
PctUsePubTrans_ = pd.qcut(murders['PctUsePubTrans'], q=4)
#Check the value counts of each bin to ensure they are balanced
PctUsePubTrans_.value_counts()

#Create bin labels
PctUsePubTrans_labels = ['0-0.4%', '0.4-1.2%', '1.2-3.3%', '3.3-55%']
#Create bins
PctUsePubTrans_bin = [-1, 0.4, 1.2, 3.3, 55]
#Add new category
murders['PctUsePubTrans_bins'] = pd.cut(murders['PctUsePubTrans'], bins=PctUsePubTrans
murders.head()
```

```
0.0
54.33
```

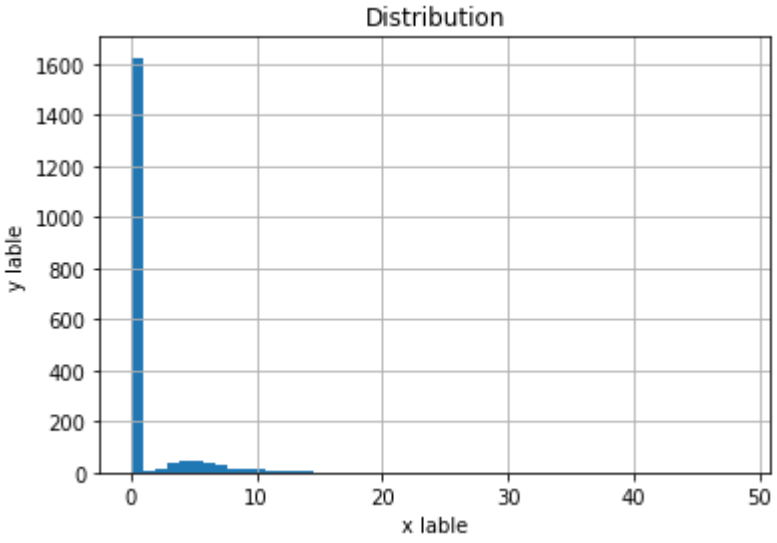
Out[ ]:

	communityname	state	population	householdsize	racepctblack	racePctWhite	racePctAsian
0	BerkeleyHeightstownship	NJ	11980	3.10	1.37	91.78	6.50
1	Marpletownship	PA	23123	2.82	0.80	95.57	3.44
2	Tigardcity	OR	29344	2.43	0.74	94.33	3.43
4	Springfieldcity	MO	140494	2.45	2.51	95.65	0.90
5	Norwoodtown	MA	28700	2.60	1.60	96.57	1.47

In [ ]:
```
murders['PctUsePubTrans_bins'].value_counts()
```

Out[ ]:
```
0-0.4%      516
3.3-55%     495
1.2-3.3%    487
0.4-1.2%    421
Name: PctUsePubTrans_bins, dtype: int64
```

In [ ]:
```
murders['LemasPctOfficDrugUn'].hist(bins=50)
plt.title('Distribution')
plt.xlabel('x lable')
plt.ylabel('y lable')
plt.show()
```



In [ ]:
```
#Check the minimum and maximum value
print(murders['LemasPctOfficDrugUn'].min())
print(murders['LemasPctOfficDrugUn'].max())

#Determine bins based on quantiles
murders['LemasPctOfficDrugUn'].describe()

#Create bin labels
PctUsePubTrans_labels = ['0%', '0.5-50%']
#Create bins
PctUsePubTrans_bin = [-1, 0.44, 60]
```

```
#Add new category
murders['PctUsePubTrans_bins'] = pd.cut(murders['PctUsePubTrans'], bins=PctUsePubTrans
murders.head()
```

```
0.0
48.44
```

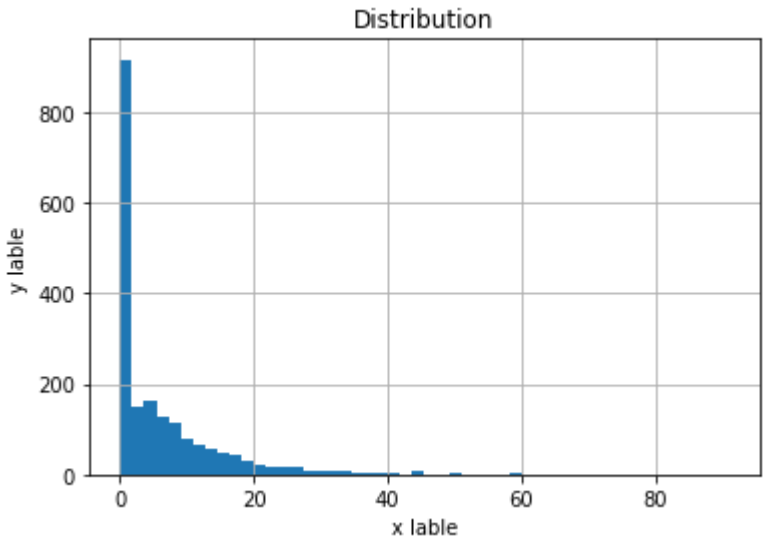
Out[ ]:

	communityname	state	population	householdsize	racepctblack	racePctWhite	racePctAsian
0	BerkeleyHeightstownship	NJ	11980	3.10	1.37	91.78	6.50
1	Marpletownship	PA	23123	2.82	0.80	95.57	3.44
2	Tigardcity	OR	29344	2.43	0.74	94.33	3.43
4	Springfieldcity	MO	140494	2.45	2.51	95.65	0.90
5	Norwoodtown	MA	28700	2.60	1.60	96.57	1.47

In [ ]:
```
murders['PctUsePubTrans_bins'].value_counts()
```

Out[ ]:
```
0.5-50%    1386
0%          533
Name: PctUsePubTrans_bins, dtype: int64
```

In [ ]:
```
murders['murdPerPop'].hist(bins=50)
plt.title('Distribution')
plt.xlabel('x lable')
plt.ylabel('y lable')
plt.show()
```



In [ ]:
```
murders_backup = murders
```

Prepare the murder dataset for CLASSIFICATION. Here, our target variable will be transformed into a binary categorical variable:

In [ ]:
```
#Check the minimum and maximum value
print(murders['murdPerPop'].min())
print(murders['murdPerPop'].max())

#Determine bins based on quantiles
murdPerPop_ = pd.qcut(murders['murdPerPop'], q=2)
#Check the value counts of each bin to ensure they are balanced
murdPerPop_.value_counts()

#Create bin labels
murdPerPop_labels = ['No', 'Yes']
#Create bins
murdPerPop_bin = [-1, 0, 100]
#Add new category
murders['murdPerPop_class_target'] = pd.cut(murders['murdPerPop'], bins=murdPerPop_bir
murders.head()
```

```
0.0
91.09
```

Out[ ]:

	communityname	state	population	householdsize	racepctblack	racePctWhite	racePctAsian
0	BerkeleyHeightstownship	NJ	11980	3.10	1.37	91.78	6.50
1	Marpletownship	PA	23123	2.82	0.80	95.57	3.44
2	Tigardcity	OR	29344	2.43	0.74	94.33	3.43
4	Springfieldcity	MO	140494	2.45	2.51	95.65	0.90
5	Norwoodtown	MA	28700	2.60	1.60	96.57	1.47

In [ ]:
```
murders['murdPerPop_class_target'].value_counts()
```

Out[ ]:
```
Yes    1043
No      876
Name: murdPerPop_class_target, dtype: int64
```

In [ ]:
```
murders_classification = murders[['state','pop_bins','householdsize_norm', 'racepctbla
                                  'racePctAsian_bins', 'racePctHisp_bins', 'agePct12t2
                                  'medIncome_bins', 'pctWFarmSelf_bins', 'pctWInvInc_n
                                  'blackPerCap_bins', 'indianPerCap_bins', 'AsianPerCa
                                  'PctPopUnderPov_bins', 'PctLess9thGrade_bins', 'PctE
                                  'PctOccupManu_norm','MalePctDivorce_norm','MalePctNe
                                  'PctImmigRecent_bins','PctRecentImmig_bins','PctPers
                                  'PctHousOccup_bins','PctVacantBoarded_bins','PctVacM
                                  'RentQrange_bins','MedRentPctHousInc_norm','MedOwnCo
                                  'PctBornSameState_bins','PctSameHouse85_norm','PctSa
                                  'PctUsePubTrans_bins', 'murdPerPop_class_target']].c

murders_classification.head()
murders_classification.info()
```

In [ ]:
```
murders_class_backup = murders_classification
murders_class_backup.info()
```

In [ ]:
```
murders_classification['state'] = murders_classification['state'].astype('category')
```

In [ ]:
```
murders_class_backup = murders_classification
```

Prepare the murder dataset for REGRESSION. Here, the target variable will be normalized numerically.

***min/max scalar function

In [ ]:
```
# apply normalization technique, using a range of [0,1]
murders['murdPerPop_reg_target'] = (murders['murdPerPop'] - murders['murdPerPop'].min(

murders.head()
```

In [ ]:
```
murders_regression = murders[['state','pop_bins','householdsize_norm', 'racepctblack_b
                              'racePctAsian_bins', 'racePctHisp_bins', 'agePct12t21_nc
                              'medIncome_bins', 'pctWFarmSelf_bins', 'pctWInvInc_norm'
                              'blackPerCap_bins', 'indianPerCap_bins', 'AsianPerCap_bi
                              'PctPopUnderPov_bins', 'PctLess9thGrade_bins', 'PctBSorN
                              'PctOccupManu_norm','MalePctDivorce_norm','MalePctNevMar
                              'PctImmigRecent_bins','PctRecentImmig_bins','PctPersOwnC
                              'PctHousOccup_bins','PctVacantBoarded_bins','PctVacMore6
                              'RentQrange_bins','MedRentPctHousInc_norm','MedOwnCostPc
                              'PctBornSameState_bins','PctSameHouse85_norm','PctSameSt
                              'PctUsePubTrans_bins', 'murdPerPop_reg_target']].copy()

murders_regression.head()
```

Out[ ]:

	state	pop_bins	householdsize_norm	racepctblack_bins	racePctWhite_bins	racePctAsian_bins	racel
0	NJ	10000-13500	0.407609	0.9-2.8%	90-96%	2.7-57.5%	
1	PA	19000-29000	0.331522	0-0.8%	90-96%	2.7-57.5%	
2	OR	29000-51500	0.225543	0-0.8%	90-96%	2.7-57.5%	
4	MO	515000-7500000	0.230978	0.9-2.8%	90-96%	0.6-1.2%	
5	MA	19000-29000	0.271739	0.9-2.8%	96-100%	1.2-2.6%	

In [ ]:
```
murders_reg_backup = murders_regression
```

In [ ]:
```
murders_regression['state'] = murders_regression['state'].astype('category')
```

Here, I want to make sure both murders and robberies datasets contain the same columns, to avoid repeating the same normalization steps.

In [ ]:
```
print(murders_copy.columns.difference(robberies.columns))
print(robberies.columns.difference(murders_copy.columns))
```

```
Index(['murdPerPop'], dtype='object')
Index(['robbbPerPop'], dtype='object')
```

Both dataframes have the exact columns, other than their target variables. Because of this, I will only normalize the robbbPerPop target variable, and create new datasets in preparation for our regression and classification models using previously normalized columns from the murders dataset.

In [ ]:
```
murders.head()
```

Out[ ]:

	communityname	state	population	householdsize	racepctblack	racePctWhite	racePctAsian
0	BerkeleyHeightstownship	NJ	11980	3.10	1.37	91.78	6.50
1	Marpletownship	PA	23123	2.82	0.80	95.57	3.44
2	Tigardcity	OR	29344	2.43	0.74	94.33	3.43
4	Springfieldcity	MO	140494	2.45	2.51	95.65	0.90
5	Norwoodtown	MA	28700	2.60	1.60	96.57	1.47

In [ ]:
```
#Create our normalized robberies dataframes

#REGRESSION DATASET
robberies_regression = murders[['state','pop_bins','householdsize_norm', 'racepctblack
                                'racePctAsian_bins', 'racePctHisp_bins', 'agePct12t21_
                                  'pctWFarmSelf_bins', 'pctWInvInc_norm', 'pctWPubAsst
                                  'indianPerCap_bins', 'AsianPerCap_bins', 'OtherPerCa
                                  'PctLess9thGrade_bins', 'PctBSorMore_bins','PctEmplM
                                  'MalePctDivorce_norm','MalePctNevMarr_norm','PctFam2
                                  'PctRecentImmig_bins','PctPersOwnOccup_norm','PctHou
                                  'PctVacantBoarded_bins','PctVacMore6Mos_norm','MedYr
                                  'MedRentPctHousInc_norm','MedOwnCostPctInc_norm','Me
                                  'PctSameHouse85_norm','PctSameState85_bins','LandAre

robberies_regression.info()
```

In [ ]:
```
robberies_regression['state'] = robberies_regression['state'].astype('category')
```

In [ ]:
```
# apply normalization technique, using a range of [0,1]
robberies_regression['robbbPerPop_reg_target'] = (robberies['robbbPerPop'] - robberies

robberies_regression.head()
```

In [ ]:
```
#CLASSIFICATION DATASET
robberies_classification = murders[['state','pop_bins','householdsize_norm', 'racepctb
                                    'racePctAsian_bins', 'racePctHisp_bins', 'agePct12
                                    'medIncome_bins', 'pctWFarmSelf_bins', 'pctWInvInc
                                    'blackPerCap_bins', 'indianPerCap_bins', 'AsianPer
                                    'PctPopUnderPov_bins', 'PctLess9thGrade_bins', 'Pc
                                    'PctOccupManu_norm','MalePctDivorce_norm','MalePct
                                    'PctImmigRecent_bins','PctRecentImmig_bins','PctPe
                                    'PctHousOccup_bins','PctVacantBoarded_bins','PctVa
                                    'RentQrange_bins','MedRentPctHousInc_norm','MedOwn
                                    'PctBornSameState_bins','PctSameHouse85_norm','Pct
                                    'PctUsePubTrans_bins']].copy()

robberies_classification.info()
```

In [ ]:
```
robberies_classification['state']=robberies_classification['state'].astype('category')
```

In [ ]:
```
#Check the minimum and maximum value
print(robberies['robbbPerPop'].min())
print(robberies['robbbPerPop'].max())

#Determine bins based on quantiles
robbbPerPop_ = pd.qcut(robberies['robbbPerPop'], q=3)
#Check the value counts of each bin to ensure they are balanced
robbbPerPop_.value_counts()

#Create bin labels
robbbPerPop_labels = ['Unlikely', 'Likely', 'Very Likely']
#Create bins
robbbPerPop_bin = [-1, 40, 145, 2500]
#Add new category
robberies_classification['robbbPerPop_class_target'] = pd.cut(robberies['robbbPerPop']
robberies_classification.head()
```

```
0.0
2264.13
```

Out[ ]:

	state	pop_bins	householdsize_norm	racepctblack_bins	racePctWhite_bins	racePctAsian_bins	racel
0	NJ	10000-13500	0.407609	0.9-2.8%	90-96%	2.7-57.5%	
1	PA	19000-29000	0.331522	0-0.8%	90-96%	2.7-57.5%	
2	OR	29000-51500	0.225543	0-0.8%	90-96%	2.7-57.5%	
4	MO	515000-7500000	0.230978	0.9-2.8%	90-96%	0.6-1.2%	
5	MA	19000-29000	0.271739	0.9-2.8%	96-100%	1.2-2.6%	

In [ ]:
```
robberies_classification['robbbPerPop_class_target'].value_counts()
```

Out[ ]:
```
Likely         656
Very Likely    642
Unlikely       621
Name: robbbPerPop_class_target, dtype: int64
```

In [ ]:
```
#MAKING COPIES OF ALL DATASETS FOR BACKUP

murd_reg_copy = murders_regression.copy()
rob_reg_copy = robberies_regression.copy()

murders_class_copy = murders_classification.copy()
rob_class_copy = robberies_classification.copy()

murders_reg_backup = murders_regression
robberies_reg_backup = robberies_regression

murdregcopy = murders_regression.copy()
```

MURDERS DATASET:

In [ ]:
```
#Make a backup copy of our dataset
murdclasscopy = murders_class_copy
```

In [ ]:
```
#Encode the cateogorical variables with LABEL ENCODING METHOD
#This code block is a test run, encoding the column 'state'
#If successful, the rest of the categorical variables will be encoded the same way

from sklearn.preprocessing import LabelEncoder

#create instance of label encoder
lab = LabelEncoder()

#perform label encoding on 'team' column
murdclasscopy['state'] = lab.fit_transform(murdclasscopy['state'])
```

In [ ]:
```
#Encode the rest of the categorical variables with for loop function

for cols in murdclasscopy.columns:
    if murdclasscopy[cols].dtype == 'category':
        murdclasscopy[cols] = lab.fit_transform(murdclasscopy[cols])
    else:
       pass
murdclasscopy.head()
```

Out[ ]:

	state	pop_bins	householdsize_norm	racepctblack_bins	racePctWhite_bins	racePctAsian_bins	racel
0	25	0	0.407609	1	2	3	
1	32	2	0.331522	0	2	3	
2	31	3	0.225543	0	2	3	
4	20	4	0.230978	1	2	1	
5	16	2	0.271739	1	3	2	

In [ ]:
```
#check info of our dataframe to ensure it is of the correct type, and that all of the

murdclasscopy.info()
```

Now we are ready to run our models. First, I will split the data into train and test sets. I will be running a LOGISTIC REGRESSION model on both the murders and robberies datasets, both preprocessed specifically for this task. Then I will running my 3 classification models: KNN, Decision Tree, and Random forest. Finally, I will be running a confusion matrix for each model. I will only compare the 3 classification models to each other, using 3 evaluation metrics: Accuracy, Precision, and Recall. The logistic regression model is for exploration purposes only, for the sake of curiosity.

Next, I will use K-Folds cross validation for my train/test split, and re-run each model for both the MURDERS and ROBBERIES dataset. I will examine the confusion matrices and run evaluation metrics again.

MURDERS DATASET:

In [ ]:
```
#Split the data into train test split
from sklearn.model_selection import train_test_split

#Split the dataset into training set and test set
#Our class column is murdPerPop_class_target, everything else will be used as features
class_murd_colname = 'murdPerPop_class_target'
feature_murd_names = murdclasscopy.columns[murdclasscopy.columns != class_murd_colname

#70% training and 30% test
x1_train, x1_test, y1_train, y1_test = train_test_split(murdclasscopy.loc[:, feature_m
                                                        murdclasscopy[class_murd_colname],
```

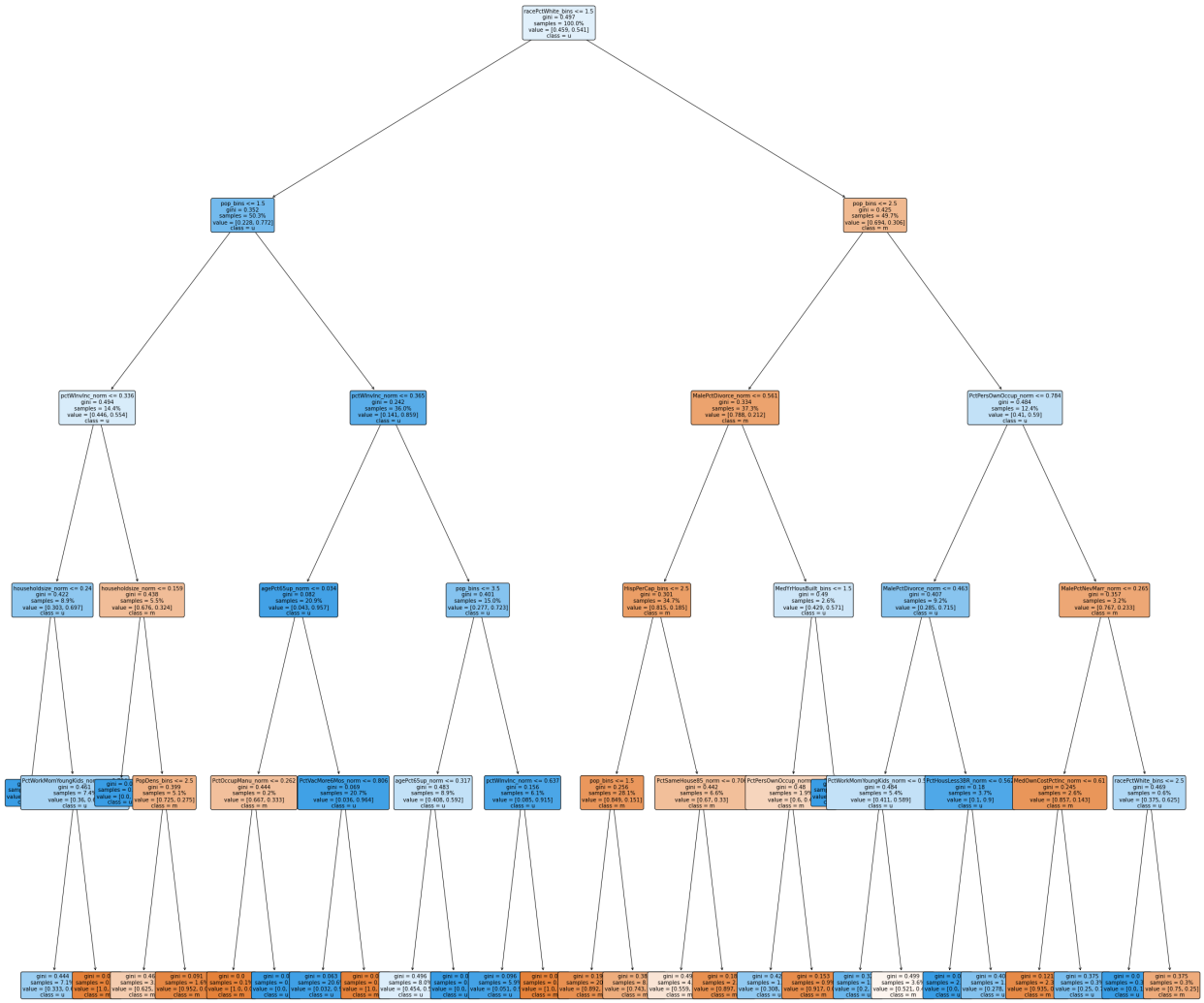
In [ ]:
```
#LOGISTIC REGRESSION
```

```
from sklearn.linear_model import LogisticRegression

logmodel = LogisticRegression(class_weight= 'balanced')
logmodel.fit(x1_train, y1_train)
predictions_log = logmodel.predict(x1_test)

predictions_log
```

In [ ]:
```
#KNN ALGORITHM

from sklearn.neighbors import KNeighborsClassifier

classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(x1_train, y1_train)

murd_pred = classifier.predict(x1_test)
murd_pred
```

In [ ]:
```
#DECISION TREE

from matplotlib import pyplot as plt
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree

#create and train the model

clf = DecisionTreeClassifier(max_depth=5, random_state=1234)
model = clf.fit(x1_train, y1_train)

#Predict the response for test dataset
murd_treepred = clf.predict(x1_test)
murd_treepred
```

In [ ]:
```
from sklearn.tree import plot_tree

#plot tree (visual representation)

features = feature_murd_names
classes = class_murd_colname

plt.figure(figsize=(40, 40))
plot_tree(clf,
          fontsize=10,
          feature_names=features,
          class_names=classes,
          rounded=True,
          filled=True,
          proportion=True);
```



In [ ]:
```
#plot tree (textual representation)

text_representation = tree.export_text(clf)
print(text_representation)
```

In [ ]:
```
#RANDOM FOREST
from sklearn.ensemble import RandomForestClassifier

randf=RandomForestClassifier()
randf.fit(x1_train, y1_train)

murd_rf_pred = randf.predict(x1_test)
murd_rf_pred
```

In [ ]:
```
#CONFUSION MATRICES

from sklearn.metrics import confusion_matrix
log_cm = confusion_matrix(y1_test, predictions_log)
knn_cm = confusion_matrix(y1_test, murd_pred)
dt_cm = confusion_matrix(y1_test, murd_treepred)
rf_cm = confusion_matrix(y1_test, murd_rf_pred)
```

```
print('LOGISTIC REGRESSION CONFUSION MATRIX')
print(log_cm)
print('KNN CONFUSION MATRIX')
print(knn_cm)
print('DECISION TREE CONFUSION MATRIX')
print(dt_cm)
print('RANDOM FOREST CONFUSION MATRIX')
print(rf_cm)
```

```
LOGISTIC REGRESSION CONFUSION MATRIX
[[206  53]
 [ 73 244]]
KNN CONFUSION MATRIX
[[183  76]
 [ 72 245]]
DECISION TREE CONFUSION MATRIX
[[197  62]
 [ 74 243]]
RANDOM FOREST CONFUSION MATRIX
[[195  64]
 [ 74 243]]
```

In [ ]:
```
#EVALUATION METRICS

log_accuracy = (log_cm[0][0] + log_cm[1][1])/(len(y1_test))
log_precision = log_cm[0][0]/(log_cm[1][0] + log_cm[0][0])
log_recall = (log_cm[0][0]/(log_cm[0][0] + log_cm[0][1]))

knn_accuracy = (knn_cm[0][0] + knn_cm[1][1])/(len(y1_test))
knn_precision = knn_cm[0][0]/(knn_cm[1][0] + knn_cm[0][0])
knn_recall = (knn_cm[0][0]/(knn_cm[0][0] + knn_cm[0][1]))

dt_accuracy = (dt_cm[0][0] + dt_cm[1][1])/(len(y1_test))
dt_precision = dt_cm[0][0]/(dt_cm[1][0] + dt_cm[0][0])
dt_recall = (dt_cm[0][0]/(dt_cm[0][0] + dt_cm[0][1]))

rf_accuracy = (rf_cm[0][0] + rf_cm[1][1])/(len(y1_test))
rf_precision = rf_cm[0][0]/(rf_cm[1][0] + rf_cm[0][0])
rf_recall = (rf_cm[0][0]/(rf_cm[0][0] + rf_cm[0][1]))

print('ACCURACY')
print('Logistic Regression: ACCURACY=', log_accuracy)
print('KNN Classifier: ACCURACY=', knn_accuracy)
print('Decision Tree: ACCURACY=', dt_accuracy)
print('Random Forest: ACCURACY=', rf_accuracy)

print('PRECISION')
print('Logistic Regression: PRECISION=', log_precision)
print('KNN Classifier: PRECISION=', knn_precision)
print('Decision Tree: PRECISION=', dt_precision)
print('Random Forest: PRECISION=', rf_accuracy)

print('RECALL')
print('Logistic Regression: RECALL=', log_recall)
print('KNN Classifier: RECALL=', knn_recall)
print('Decision Tree: RECALL=', dt_recall)
print('Random Forest: RECALL=', rf_accuracy)
```

```
ACCURACY
Logistic Regression: ACCURACY= 0.78125
KNN Classifier: ACCURACY= 0.7430555555555556
Decision Tree: ACCURACY= 0.7638888888888888
Random Forest: ACCURACY= 0.7604166666666666
PRECISION
Logistic Regression: PRECISION= 0.7383512544802867
KNN Classifier: PRECISION= 0.7176470588235294
Decision Tree: PRECISION= 0.7269372693726938
Random Forest: PRECISION= 0.7604166666666666
RECALL
Logistic Regression: RECALL= 0.7953667953667953
KNN Classifier: RECALL= 0.7065637065637066
Decision Tree: RECALL= 0.7606177606177607
Random Forest: RECALL= 0.7604166666666666
```

MURDERS: K-FOLD CROSS VALIDATION

In [ ]:
```
# define dataset
X, y = murdclasscopy.loc[:, feature_murd_names], murdclasscopy[class_murd_colname]
# summarize the dataset
print(X.shape, y.shape)
```

```
(1919, 50) (1919,)
```

In [ ]:
```
from numpy import mean
from numpy import std
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression

# prepare the cross-validation procedure
cv = KFold(n_splits=10, random_state=42, shuffle=True)
```

In [ ]:
```
# evaluate a logistic regression model using k-fold cross-validation

# create model
log_model = LogisticRegression()
# evaluate model
log_scores_acc = cross_val_score(log_model, X, y, scoring='accuracy', cv=cv, n_jobs=-1
log_scores_pre = cross_val_score(log_model, X, y, scoring='precision', cv=cv, n_jobs=-
log_scores_re = cross_val_score(log_model, X, y, scoring='recall', cv=cv, n_jobs=-1)
# report performance
print('Accuracy: %.3f (%.3f)' % (mean(log_scores_acc), std(log_scores_acc)))
print('Precision: %.3f (%.3f)' % (mean(log_scores_pre), std(log_scores_pre)))
print('Recall: %.3f (%.3f)' % (mean(log_scores_re), std(log_scores_re)))
```

```
Accuracy: 0.776 (0.029)
Precision: 0.785 (0.033)
Recall: 0.806 (0.040)
```

In [ ]:
```
# evaluate a knn model using k-fold cross-validation

# create model
knn_model = KNeighborsClassifier()
# evaluate model
knn_scores_acc = cross_val_score(knn_model, X, y, scoring='accuracy', cv=cv, n_jobs=-1
knn_scores_pre = cross_val_score(knn_model, X, y, scoring='precision', cv=cv, n_jobs=-
knn_scores_re = cross_val_score(knn_model, X, y, scoring='recall', cv=cv, n_jobs=-1)
# report performance
```

```
print('Accuracy: %.3f (%.3f)' % (mean(knn_scores_acc), std(knn_scores_acc)))
print('Precision: %.3f (%.3f)' % (mean(knn_scores_pre), std(knn_scores_pre)))
print('Recall: %.3f (%.3f)' % (mean(knn_scores_re), std(knn_scores_re)))
```

```
Accuracy: 0.752 (0.028)
Precision: 0.759 (0.030)
Recall: 0.795 (0.036)
```

In [ ]:
```
# evaluate a decision tree model using k-fold cross-validation

# create model
dt_model = DecisionTreeClassifier(max_depth=5)
# evaluate model
dt_scores_acc = cross_val_score(dt_model, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
dt_scores_pre = cross_val_score(dt_model, X, y, scoring='precision', cv=cv, n_jobs=-1)
dt_scores_re = cross_val_score(dt_model, X, y, scoring='recall', cv=cv, n_jobs=-1)
# report performance
print('Accuracy: %.3f (%.3f)' % (mean(dt_scores_acc), std(dt_scores_acc)))
print('Precision: %.3f (%.3f)' % (mean(dt_scores_pre), std(dt_scores_pre)))
print('Recall: %.3f (%.3f)' % (mean(dt_scores_re), std(dt_scores_re)))
```

```
Accuracy: 0.727 (0.029)
Precision: 0.767 (0.042)
Recall: 0.713 (0.040)
```

In [ ]:
```
# evaluate a random forest model using k-fold cross-validation

# create model
rf_model = RandomForestClassifier()
# evaluate model
rf_scores_acc = cross_val_score(rf_model, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
rf_scores_pre = cross_val_score(rf_model, X, y, scoring='precision', cv=cv, n_jobs=-1)
rf_scores_re = cross_val_score(rf_model, X, y, scoring='recall', cv=cv, n_jobs=-1)
# report performance
print('Accuracy: %.3f (%.3f)' % (mean(rf_scores_acc), std(rf_scores_acc)))
print('Precision: %.3f (%.3f)' % (mean(rf_scores_pre), std(rf_scores_pre)))
print('Recall: %.3f (%.3f)' % (mean(rf_scores_re), std(rf_scores_re)))
```

```
Accuracy: 0.768 (0.028)
Precision: 0.794 (0.030)
Recall: 0.786 (0.032)
```

ROBBERIES DATASET:

In [ ]:
```
#Create a copy of the dataset

robclasscopy = rob_class_copy.copy()
robclasscopy.head()
```

Out[ ]:

	state	pop_bins	householdsize_norm	racepctblack_bins	racePctWhite_bins	racePctAsian_bins	racel
0	NJ	10000-13500	0.407609	0.9-2.8%	90-96%	2.7-57.5%	
1	PA	19000-29000	0.331522	0-0.8%	90-96%	2.7-57.5%	
2	OR	29000-51500	0.225543	0-0.8%	90-96%	2.7-57.5%	
4	MO	515000-7500000	0.230978	0.9-2.8%	90-96%	0.6-1.2%	
5	MA	19000-29000	0.271739	0.9-2.8%	96-100%	1.2-2.6%	

In [ ]:
```
#Encode the cateogorical variables with LABEL ENCODING METHOD
#This code block is a test run, encoding the column 'state'
#If successful, the rest of the categorical variables will be encoded the same way

#perform label encoding on 'team' column
robclasscopy['state'] = lab.fit_transform(robclasscopy['state'])

robclasscopy.head(3)
```

Out[ ]:

	state	pop_bins	householdsize_norm	racepctblack_bins	racePctWhite_bins	racePctAsian_bins	racel
0	25	10000-13500	0.407609	0.9-2.8%	90-96%	2.7-57.5%	
1	32	19000-29000	0.331522	0-0.8%	90-96%	2.7-57.5%	
2	31	29000-51500	0.225543	0-0.8%	90-96%	2.7-57.5%	

In [ ]:
```
#Encode the rest of the categorical variables with for loop function

for r_cols in robclasscopy.columns:
    if robclasscopy[r_cols].dtype == 'category':
        robclasscopy[r_cols] = lab.fit_transform(robclasscopy[r_cols])
    else:
       pass
robclasscopy.head()
```

Out[ ]:

	state	pop_bins	householdsize_norm	racepctblack_bins	racePctWhite_bins	racePctAsian_bins	racel
0	25	0	0.407609	1	2	3	
1	32	2	0.331522	0	2	3	
2	31	3	0.225543	0	2	3	
4	20	4	0.230978	1	2	1	
5	16	2	0.271739	1	3	2	

In [ ]:
```
#check info of our dataframe to ensure it is of the correct type, and that all of the

robclasscopy.info()
```

In [ ]:
```
#Split the data into train test split
from sklearn.model_selection import train_test_split

#Split the dataset into training set and test set
#Our class column is murdPerPop_class_target, everything else will be used as features
class_rob_colname = 'robbbPerPop_class_target'
feature_rob_names = robclasscopy.columns[robclasscopy.columns != class_rob_colname]

#70% training and 30% test
x2_train, x2_test, y2_train, y2_test = train_test_split(robclasscopy.loc[:, feature_rc
                                                        robclasscopy[class_rob_colname], t
```

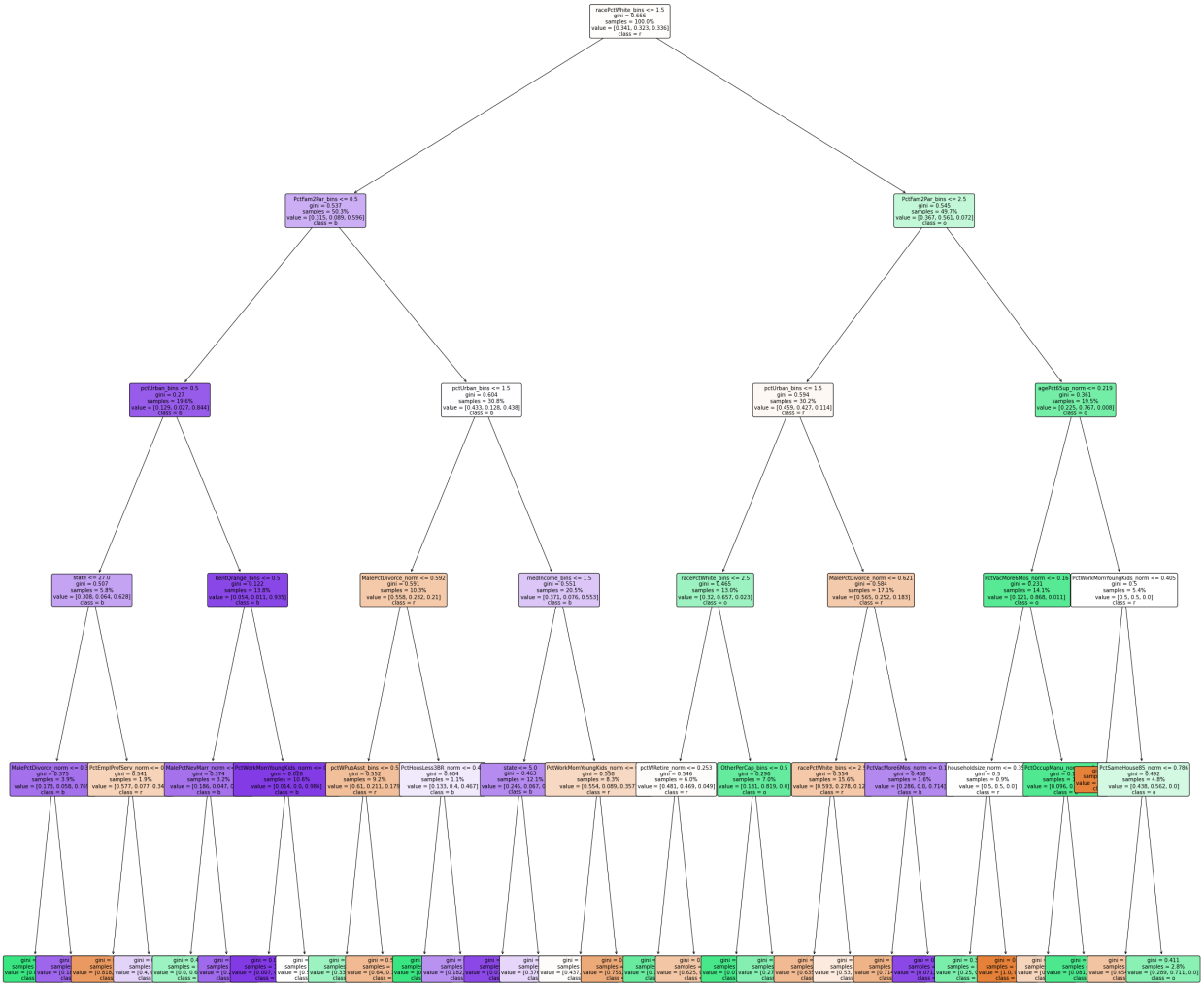
In [ ]:
```
#LOGISTIC REGRESSION

from sklearn.linear_model import LogisticRegression

rob_logmodel = LogisticRegression(class_weight= 'balanced')
logmodel.fit(x2_train, y2_train)
rob_pred_log = logmodel.predict(x2_test)

rob_pred_log
```

In [ ]:
```
#KNN ALGORITHM

from sklearn.neighbors import KNeighborsClassifier

rob_classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(x2_train, y2_train)

rob_pred = classifier.predict(x2_test)
rob_pred
```

In [ ]:
```
#DECISION TREE

from matplotlib import pyplot as plt
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree

#create and train the model

r_clf = DecisionTreeClassifier(max_depth=5, random_state=1234)
```

```
r_model = r_clf.fit(x2_train, y2_train)

#Predict the response for test dataset
rob_treepred = r_clf.predict(x2_test)
rob_treepred
```

In [ ]:
```
from sklearn.tree import plot_tree

#plot tree (visual representation)

r_features = feature_rob_names
r_classes = class_rob_colname

plt.figure(figsize=(40, 40))
plot_tree(r_clf,
          fontsize=10,
          feature_names=r_features,
          class_names=r_classes,
          rounded=True,
          filled=True,
          proportion=True);
```



In [ ]:
```
#plot tree (textual representation)

text_representation = tree.export_text(r_clf)
print(text_representation)
```

In [ ]:
```
#RANDOM FOREST
from sklearn.ensemble import RandomForestClassifier

r_randf=RandomForestClassifier()
r_randf.fit(x2_train, y2_train)

rob_rf_pred = r_randf.predict(x1_test)
rob_rf_pred
```

In [ ]:
```
#CONFUSION MATRICES

rlog_cm = confusion_matrix(y2_test, rob_pred_log)
rknn_cm = confusion_matrix(y2_test, rob_pred)
rdt_cm = confusion_matrix(y2_test, rob_treepred)
rrf_cm = confusion_matrix(y2_test, rob_rf_pred)

print('LOGISTIC REGRESSION CONFUSION MATRIX')
print(rlog_cm)
print('KNN CONFUSION MATRIX')
print(rknn_cm)
print('DECISION TREE CONFUSION MATRIX')
print(rdt_cm)
print('RANDOM FOREST CONFUSION MATRIX')
print(rrf_cm)
```

```
LOGISTIC REGRESSION CONFUSION MATRIX
[[112  52  34]
 [ 43 141   3]
 [ 31   4 156]]
KNN CONFUSION MATRIX
[[100  56  42]
 [ 55 125   7]
 [ 37   6 148]]
DECISION TREE CONFUSION MATRIX
[[120  43  35]
 [ 64 119   4]
 [ 32   5 154]]
RANDOM FOREST CONFUSION MATRIX
[[127  41  30]
 [ 39 147   1]
 [ 27   2 162]]
```

In [ ]:
```
#EVALUATION METRICS
from sklearn import metrics

rlog_accuracy = (metrics.accuracy_score(y2_test, rob_pred_log))
rlog_precision = (metrics.precision_score(y2_test, rob_pred_log, average='macro'))
rlog_recall = (metrics.recall_score(y2_test, rob_pred_log, average='macro'))

rknn_accuracy = (metrics.accuracy_score(y2_test, rob_pred))
rknn_precision = (metrics.precision_score(y2_test, rob_pred, average='macro'))
rknn_recall = (metrics.recall_score(y2_test, rob_pred, average='macro'))

rdt_accuracy = (metrics.accuracy_score(y2_test, rob_treepred))
```

```
rdt_precision = (metrics.precision_score(y2_test, rob_treepred, average='macro'))
rdt_recall = (metrics.recall_score(y2_test, rob_treepred, average='macro'))


rrf_accuracy = (metrics.accuracy_score(y2_test, rob_rf_pred))
rrf_precision = (metrics.precision_score(y2_test, rob_rf_pred, average='macro'))
rrf_recall = (metrics.recall_score(y2_test, rob_rf_pred, average='macro'))

print('ACCURACY')
print('Logistic Regression: ACCURACY=', rlog_accuracy)
print('KNN Classifier: ACCURACY=', rknn_accuracy)
print('Decision Tree: ACCURACY=', rdt_accuracy)
print('Random Forest: ACCURACY=', rrf_accuracy)

print('PRECISION')
print('Logistic Regression: PRECISION=', rlog_precision)
print('KNN Classifier: PRECISION=', rknn_precision)
print('Decision Tree: PRECISION=', rdt_precision)
print('Random Forest: PRECISION=', rrf_accuracy)

print('RECALL')
print('Logistic Regression: RECALL=', rlog_recall)
print('KNN Classifier: RECALL=', rknn_recall)
print('Decision Tree: RECALL=', rdt_recall)
print('Random Forest: RECALL=', rrf_accuracy)
```

```
ACCURACY
Logistic Regression: ACCURACY= 0.7100694444444444
KNN Classifier: ACCURACY= 0.6475694444444444
Decision Tree: ACCURACY= 0.6822916666666666
Random Forest: ACCURACY= 0.7569444444444444
PRECISION
Logistic Regression: PRECISION= 0.7087255778946533
KNN Classifier: PRECISION= 0.6468505222424303
Decision Tree: PRECISION= 0.688685955664951
Random Forest: PRECISION= 0.7569444444444444
RECALL
Logistic Regression: RECALL= 0.7121403958484341
KNN Classifier: RECALL= 0.6494562709530373
Decision Tree: RECALL= 0.6829023216457771
Random Forest: RECALL= 0.7569444444444444
```

ROBBERIES: K-FOLD CROSS VALIDATION

In [ ]:
```
# define dataset
X1, y1 = robclasscopy.loc[:, feature_rob_names], robclasscopy[class_rob_colname]
# summarize the dataset
print(X1.shape, y1.shape)
```

```
(1919, 50) (1919,)
```

In [ ]:
```
# evaluate a logistic regression model using k-fold cross-validation

# prepare the cross-validation procedure
cv = KFold(n_splits=10, random_state=42, shuffle=True)
# create model
log_model = LogisticRegression()
# evaluate model
rlog_scores_acc = cross_val_score(log_model, X1, y1, scoring='accuracy', cv=cv, n_jobs
rlog_scores_pre = cross_val_score(log_model, X1, y1, scoring='precision', cv=cv, n_job
```

```
rlog_scores_re = cross_val_score(log_model, X1, y1, scoring='recall', cv=cv, n_jobs=-1
# report performance
print('Accuracy: %.3f (%.3f)' % (mean(rlog_scores_acc), std(rlog_scores_acc)))
print('Precision:',(metrics.precision_score(y2_test, rob_pred_log, average = 'macro'))
print('Recall:',(metrics.recall_score(y2_test, rob_pred_log, average = 'macro')))
```

```
Accuracy: 0.705 (0.042)
Precision: 0.7087255778946533
Recall: 0.7121403958484341
```

In [ ]:
```
# evaluate a knn model using k-fold cross-validation

# create model
knn_model = KNeighborsClassifier()
# evaluate model
rknn_scores_acc = cross_val_score(knn_model, X1, y1, scoring='accuracy', cv=cv, n_jobs
rknn_scores_pre = cross_val_score(knn_model, X1, y1, scoring='precision', cv=cv, n_job
rknn_scores_re = cross_val_score(knn_model, X1, y1, scoring='recall', cv=cv, n_jobs=-1
# report performance
print('Accuracy: %.3f (%.3f)' % (mean(rknn_scores_acc), std(rknn_scores_acc)))
print('Precision:',(metrics.precision_score(y2_test, rob_pred, average = 'macro')))
print('Recall:',(metrics.recall_score(y2_test, rob_pred, average = 'macro')))
```

```
Accuracy: 0.646 (0.031)
Precision: 0.6468505222424303
Recall: 0.6494562709530373
```

In [ ]:
```
# evaluate a decision tree model using k-fold cross-validation

# create model
dt_model = DecisionTreeClassifier(max_depth=5)
# evaluate model
rdt_scores_acc = cross_val_score(dt_model, X1, y1, scoring='accuracy', cv=cv, n_jobs=-
rdt_scores_pre = cross_val_score(dt_model, X1, y1, scoring='precision', cv=cv, n_jobs=
rdt_scores_re = cross_val_score(dt_model, X1, y1, scoring='recall', cv=cv, n_jobs=-1)
# report performance
print('Accuracy: %.3f (%.3f)' % (mean(rdt_scores_acc), std(rdt_scores_acc)))
print('Precision:',(metrics.precision_score(y2_test, rob_treepred, average = 'macro'))
print('Recall:', (metrics.recall_score(y2_test, rob_treepred, average = 'macro')))
```

```
Accuracy: 0.663 (0.027)
Precision: 0.688685955664951
Recall: 0.6829023216457771
```

In [ ]:
```
# evaluate a random forest model using k-fold cross-validation

# create model
rf_model = RandomForestClassifier()
# evaluate model
rrf_scores_acc = cross_val_score(rf_model, X1, y1, scoring='accuracy', cv=cv, n_jobs=-
rrf_scores_pre = cross_val_score(rf_model, X1, y1, scoring='precision', cv=cv, n_jobs=
rrf_scores_re = cross_val_score(rf_model, X1, y1, scoring='recall', cv=cv, n_jobs=-1)
# report performance
print('Accuracy: %.3f (%.3f)' % (mean(rrf_scores_acc),std(rdt_scores_acc)))
print('Precision:',(metrics.precision_score(y2_test, rob_rf_pred, average = 'macro')))
print('Recall:', (metrics.recall_score(y2_test, rob_rf_pred, average = 'macro')))
```

```
Accuracy: 0.714 (0.026)
Precision: 0.7570311789837287
Recall: 0.7585593124552164
```