State and County Social Vulnerability Analysis Script (2024)

by Wenhao Wu, 2024

This is a data analysis python script that pulls various data from ACS via API for a statewide Social Vulnerability Analysis at the census tract level, with Principal Component Analysis (PCA) and Machine Learning Predictions. The outputs are a set of statewide GIS shapefiles and spreadsheets that can be used for specific analysis at the jurisdiction or regional levels. More information of this standardized workflow can be found here: W:\Electronic Library\Social Vulnerability to Climate

Analysis Outline:

1. Set up and Data Processing

2. Exploratory Data Analysis

3. Principal Component Analysis (PCA)

4. optional: Machine Learning Predictive Analysis

1. Set up and Data Processing

In [2]:
```
# import selenium
# selenium.__version__
import os
import plotly.io as pio
from pathlib import Path

import geopandas as gpd ##some issues with the 'geopandas' conda env, may need to switch to a separate env!
from geopandas import GeoDataFrame
# from osgeo import gdal, ogr, osr
# from fiona.ogrext import Iterator, ItemsIterator, KeysIterator

import math
import numpy as np
import sys
import pandas as pd
import matplotlib.pyplot as plt

#modify column display settings
pd.set_option('display.max_columns',None)
```

```
#api data
import requests

#seaborn for plotting
import seaborn as sb

#mapping
import contextily as cx
import folium

#scikit learn
from sklearn.datasets import make_regression
from sklearn.metrics import pairwise
from sklearn.decomposition import PCA
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
```

Fetch data from USCS API

In [3]:
```
### ACS API to import ACS data ###
#Common demographics + Sensitive Pop (draft) by tract:
#ref: https://levelup.gitconnected.com/how-to-get-total-population-from-the-census-api-using-python-bbf23758bfa7
#https://atcoordinates.info/2019/09/24/examples-of-using-the-census-bureaus-api-with-python/

# direct API call: https://api.census.gov/data/2021/acs/acs5?get=TRACT,NAME,B01001_001E,B01001_003E,B01001_027E,C18120_004E,C181
# all variables:

year = '2022'

#the number of variables cannot exceed 50
variables = 'TRACT,B01001_001E,B01001_003E,B01001_027E,\
C18120_004E,C18120_007E,C18120_010E,C18120_001E,C24050_002E,C24050_001E,\
B06009_002E,B06009_001E,B17001_002E,B17001_001E,C18120_006E,C18120_002E,\
B25070_010E,B25070_001E,B25091_011E,B25091_022E,B25091_001E,\
B02001_001E,B02001_002E,B02001_003E,B02001_004E,B02001_005E,B02001_006E,B02001_007E,B02001_008E,\
B03002_001E,B03002_002E,B03002_003E,B03002_004E,B03002_005E,B03002_006E,B03002_007E,B03002_008E,B03002_009E,B03002_012E'

api_key = '293a71b49d3b2b1f7e5ac960faa522a51cf6cc24' ##actually no need of api key for a public data request
url = 'https://api.census.gov/data/{0}/acs/acs5?get=NAME,{1}&for=tract:*&in=state:06&key={2}'.format(year,variables,api_key)

response = requests.request("GET",url)
# print(response.text) #this line exceeds the output data limit
```

```
#parse to data frame
data = response.json()
df0 = pd.DataFrame(data[1:],columns=data[0])
df0['GEOID_2'] = df0['state'] + df0['county'] + df0['TRACT']
df0.columns
```

Out[3]:
```
Index(['NAME', 'TRACT', 'B01001_001E', 'B01001_003E', 'B01001_027E',
       'C18120_004E', 'C18120_007E', 'C18120_010E', 'C18120_001E',
       'C24050_002E', 'C24050_001E', 'B06009_002E', 'B06009_001E',
       'B17001_002E', 'B17001_001E', 'C18120_006E', 'C18120_002E',
       'B25070_010E', 'B25070_001E', 'B25091_011E', 'B25091_022E',
       'B25091_001E', 'B02001_001E', 'B02001_002E', 'B02001_003E',
       'B02001_004E', 'B02001_005E', 'B02001_006E', 'B02001_007E',
       'B02001_008E', 'B03002_001E', 'B03002_002E', 'B03002_003E',
       'B03002_004E', 'B03002_005E', 'B03002_006E', 'B03002_007E',
       'B03002_008E', 'B03002_009E', 'B03002_012E', 'state', 'county', 'tract',
       'GEOID_2'],
      dtype='object')
```

In [4]:
```
#Get adaptive capacity variables - if combined with above will cause error...
year = '2022'

variables1 = 'TRACT,B01001_020E,B01001_021E,B01001_022E,B01001_023E,B01001_024E,B01001_025E,\
B01001_044E,B01001_045E,B01001_046E,B01001_047E,B01001_048E,B01001_049E,\
B25003_001E,B25003_003E,B25034_001E,B25034_007E,B25034_008E,B25034_009E,B25034_010E,B25034_011E,\
B25024_001E,B25024_010E,B25044_001E,B25044_003E,B25044_010E,B28002_001E,B28002_013E,\
C16001_001E,C16001_005E,C16001_008E,C16001_011E,C16001_014E,C16001_017E,\
C16001_020E,C16001_023E,C16001_026E,C16001_029E,C16001_032E,C16001_035E,C16001_038E'

api_key = '293a71b49d3b2b1f7e5ac960faa522a51cf6cc24' ##actually no need of api key for a public data request
url1 = 'https://api.census.gov/data/{0}/acs/acs5?get=NAME,{1}&for=tract:*&in=state:06&key={2}'.format(year,variables1,api_key)

response1 = requests.request("GET",url1)
# print(response.text)

#parse to data frame
data1 = response1.json()
df1 = pd.DataFrame(data1[1:],columns=data1[0])
df1['GEOID_2'] = df1['state'] + df1['county'] + df1['TRACT']
df1 = df1.drop(columns=['NAME','state','county','tract','TRACT'])
df1.columns
```

Out[4]:
```
Index(['B01001_020E', 'B01001_021E', 'B01001_022E', 'B01001_023E',
       'B01001_024E', 'B01001_025E', 'B01001_044E', 'B01001_045E',
       'B01001_046E', 'B01001_047E', 'B01001_048E', 'B01001_049E',
       'B25003_001E', 'B25003_003E', 'B25034_001E', 'B25034_007E',
       'B25034_008E', 'B25034_009E', 'B25034_010E', 'B25034_011E',
       'B25024_001E', 'B25024_010E', 'B25044_001E', 'B25044_003E',
       'B25044_010E', 'B28002_001E', 'B28002_013E', 'C16001_001E',
       'C16001_005E', 'C16001_008E', 'C16001_011E', 'C16001_014E',
       'C16001_017E', 'C16001_020E', 'C16001_023E', 'C16001_026E',
       'C16001_029E', 'C16001_032E', 'C16001_035E', 'C16001_038E', 'GEOID_2'],
      dtype='object')
```

In [5]:
```
#join the two dfs
df = df0.merge(df1,how='inner',on='GEOID_2')
df.head(3)
```

Out[5]:

	NAME	TRACT	B01001_001E	B01001_003E	B01001_027E	C18120_004E	C18120_007E	C18120_010E	C18120_001E	C24050_002E	C24050_001E
0	Census Tract 4001; Alameda County; California	400100	3269	72	61	53	9	36	1724	9	1641
1	Census Tract 4002; Alameda County; California	400200	2147	83	86	23	0	23	1244	0	1124
2	Census Tract 4003; Alameda County; California	400300	5619	34	93	59	96	28	3761	10	3350

In [6]:
```
df.columns
```

Out[6]:
```
Index(['NAME', 'TRACT', 'B01001_001E', 'B01001_003E', 'B01001_027E',
       'C18120_004E', 'C18120_007E', 'C18120_010E', 'C18120_001E',
       'C24050_002E', 'C24050_001E', 'B06009_002E', 'B06009_001E',
       'B17001_002E', 'B17001_001E', 'C18120_006E', 'C18120_002E',
       'B25070_010E', 'B25070_001E', 'B25091_011E', 'B25091_022E',
       'B25091_001E', 'B02001_001E', 'B02001_002E', 'B02001_003E',
       'B02001_004E', 'B02001_005E', 'B02001_006E', 'B02001_007E',
       'B02001_008E', 'B03002_001E', 'B03002_002E', 'B03002_003E',
       'B03002_004E', 'B03002_005E', 'B03002_006E', 'B03002_007E',
       'B03002_008E', 'B03002_009E', 'B03002_012E', 'state', 'county', 'tract',
       'GEOID_2', 'B01001_020E', 'B01001_021E', 'B01001_022E', 'B01001_023E',
       'B01001_024E', 'B01001_025E', 'B01001_044E', 'B01001_045E',
       'B01001_046E', 'B01001_047E', 'B01001_048E', 'B01001_049E',
       'B25003_001E', 'B25003_003E', 'B25034_001E', 'B25034_007E',
       'B25034_008E', 'B25034_009E', 'B25034_010E', 'B25034_011E',
       'B25024_001E', 'B25024_010E', 'B25044_001E', 'B25044_003E',
       'B25044_010E', 'B28002_001E', 'B28002_013E', 'C16001_001E',
       'C16001_005E', 'C16001_008E', 'C16001_011E', 'C16001_014E',
       'C16001_017E', 'C16001_020E', 'C16001_023E', 'C16001_026E',
       'C16001_029E', 'C16001_032E', 'C16001_035E', 'C16001_038E'],
      dtype='object')
```

In [7]:
```
#rename the columns
df = df.rename(columns={
                        #Common sociodemographics variables
                        'B01001_001E':'Total Pop',
                        'B01001_003E':'Male under 5',
                        'B01001_027E':'Female under 5',
                        'C18120_004E':'Employeed with Disability',
                        'C18120_007E':'Unemployeed with Disability',
                        'C18120_010E':'Not in Labor with Disability',
                        'C18120_001E':'Total Employment by Disability',
                        'C24050_002E':'Agri, Frst, Fish and Mining Workers',
                        'C24050_001E':'Total Workers',

                        'B02001_001E':'Total Pop by Race', ##race and ethnicity variables
                        'B02001_002E':'White alone',
                        'B02001_003E':'Black alone',
                        'B02001_004E':'AIAN alone',
                        'B02001_005E':'Asian alone',
                        'B02001_006E':'NHPI alone',
                        'B02001_007E':'Other race alone',
                        'B02001_008E':'Two or more races',
                        'B03002_001E':'Total Pop by Hispanic Origin',
                        'B03002_002E':'Not Hispanic or Latino',
                        'B03002_003E':'Non-Hispanic White alone',
                        'B03002_004E':'Non-Hispanic Black alone',
                        'B03002_005E':'Non-Hispanic AIAN alone',
                        'B03002_006E':'Non-Hispanic Asian alone',
```

```
                        'B03002_007E':'Non-Hispanic NHPI alone',
                        'B03002_008E':'Non-Hispanic other race alone',
                        'B03002_009E':'Non-Hispanic two or more races',
                        'B03002_012E':'Hispanic or Latino',


                        #Sensitivity(CalEnviroScreen) variables
                        'B06009_002E':'Less than High School',
                        'B06009_001E':'Total Education',
                        'B17001_002E':'Income Below Poverty Level',
                        'B17001_001E':'Total Poverty Status',
                        'C18120_006E':'Unemployeed in Labor',
                        'C18120_002E':'Total In Labor',
                        'B25070_010E':'Gross Rent 50pct or more of Income',
                        'B25070_001E':'Total Gross Rent as pct of Income',
                        'B25091_011E':'Mortgaged Owner Cost 50pct or more of Income',
                        'B25091_022E':'Unmortgaged Owner Cost 50pct or more of Income',
                        'B25091_001E':'Total Owner Costs as pct of Income',
                        ##TBD: linguistic isolation


                        #Adaptive capacity variables
                        'B25003_001E':'Total by Tenure',
                        'B25003_003E':'Renter-Occupied Housing',
                        'B25034_001E':'Total by Year-built',
                        'B25034_007E':'Year-built 1970-79',
                        'B25034_008E':'Year-built 1960-69',
                        'B25034_009E':'Year-built 1950-59',
                        'B25034_010E':'Year-built 1940-49',
                        'B25034_011E':'Year-built 1939 or earlier',
                        'B25024_001E':'Total Housing by Type',
                        'B25024_010E':'Mobile Homes',
                        'B25044_001E':'Total Tenure by Vehicles',
                        'B25044_003E':'Owner No Vehicle',
                        'B25044_010E':'Renter No Vehicle',
                        ##TBD: housing cost burden - included above already?
                        'B28002_001E':'Total Internet Access',
                        'B28002_013E':'No Internet Access'

                       })
df.head(3)
# df.columns
# df.dtypes
```

Out[7]:

	NAME	TRACT	Total Pop	Male under 5	Female under 5	Employeed with Disability	Unemployeed with Disability	Not in Labor with Disability	Total Employment by Disability	Agri, Frst, Fish and Mining Workers	Total Workers	Less than High School	Total Education	Income Below Poverty Level	Total Poverty Status	Un
0	Census Tract 4001; Alameda County; California	400100	3269	72	61	53	9	36	1724	9	1641	55	2522	142	3248	
1	Census Tract 4002; Alameda County; California	400200	2147	83	86	23	0	23	1244	0	1124	50	1740	161	2147	
2	Census Tract 4003; Alameda County; California	400300	5619	34	93	59	96	28	3761	10	3350	181	4514	396	5592	

In [8]:
```
#convert data types for pct calculation
df[df.columns[1:]] = df[df.columns[1:]].astype(float)
df['GEOID_2'] = df['GEOID_2'].astype(str).str[0:10] #make sure to incude the .str!!!
# df.dtypes
```

In [9]:
```
# pct columns calculation
df['pct over 65'] = (df['B01001_020E'] + df['B01001_021E'] + df['B01001_022E'] + df['B01001_023E'] + df['B01001_024E'] + df['B01
                     + df['B01001_044E'] + df['B01001_045E'] + df['B01001_046E'] + df['B01001_047E'] + df['B01001_048E'] + df['B
df['pct under 5'] = (df['Male under 5']+df['Female under 5'])/df['Total Pop']
df['pct disability'] = (df['Employeed with Disability']+df['Unemployeed with Disability']+df['Not in Labor with Disability'])/df
df['pct ag workers'] = df['Agri, Frst, Fish and Mining Workers']/df['Total Workers']
df['pct no high school'] = df['Less than High School']/df['Total Education']
df['pct of limited English'] = (df['C16001_005E'] + df['C16001_008E'] + df['C16001_011E'] +
                                df['C16001_014E'] + df['C16001_017E'] + df['C16001_020E'] +
                                df['C16001_023E'] + df['C16001_026E'] + df['C16001_029E'] +
                                df['C16001_032E'] + df['C16001_035E'] + df['C16001_038E'])/df['C16001_001E']

df['pct in poverty'] = df['Income Below Poverty Level']/df['Total Poverty Status']
df['pct unemployment'] = df['Unemployeed in Labor']/df['Total In Labor']
df['pct renter severely burdened'] = df['Gross Rent 50pct or more of Income']/df['Total Gross Rent as pct of Income']
df['pct owner severely burdened'] = (df['Mortgaged Owner Cost 50pct or more of Income']+df['Unmortgaged Owner Cost 50pct or more

df['pct non-hispanic white'] = df['Non-Hispanic White alone']/df['Total Pop by Hispanic Origin']
```

```
df['pct non-hispanic black'] = df['Non-Hispanic Black alone']/df['Total Pop by Hispanic Origin']
df['pct non-hispanic aian'] = df['Non-Hispanic AIAN alone']/df['Total Pop by Hispanic Origin']
df['pct non-hispanic asian'] = df['Non-Hispanic Asian alone']/df['Total Pop by Hispanic Origin']
df['pct non-hispanic nhpi'] = df['Non-Hispanic NHPI alone']/df['Total Pop by Hispanic Origin']
df['pct non-hispanic other'] = df['Non-Hispanic other race alone']/df['Total Pop by Hispanic Origin']
df['pct non-hispanic two'] = df['Non-Hispanic two or more races']/df['Total Pop by Hispanic Origin']
df['pct hispanic'] = df['Hispanic or Latino']/df['Total Pop by Hispanic Origin']

df['pct renter'] = df['Renter-Occupied Housing']/df['Total by Tenure']
df['pct pre-1980 housing'] = (df['Year-built 1970-79']+df['Year-built 1960-69']+df['Year-built 1950-59']+df['Year-built 1940-49'
df['pct mobile homes'] = df['Mobile Homes']/df['Total Housing by Type']
df['pct no vehicle'] = (df['Owner No Vehicle']+df['Renter No Vehicle'])/df['Total Tenure by Vehicles']
df['pct no internet'] = df['No Internet Access']/df['Total Internet Access']

df['GEOID_Copy'] = df['GEOID_2']

df.head(3)
```

```
/var/folders/g6/p9107ts97cb3b4gk5zy0yfrm0000gn/T/ipykernel_7600/3035970618.py:24: PerformanceWarning: DataFrame is highly fragme
nted.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all column
s at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`
  df['pct non-hispanic two'] = df['Non-Hispanic two or more races']/df['Total Pop by Hispanic Origin']
/var/folders/g6/p9107ts97cb3b4gk5zy0yfrm0000gn/T/ipykernel_7600/3035970618.py:25: PerformanceWarning: DataFrame is highly fragme
nted.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all column
s at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`
  df['pct hispanic'] = df['Hispanic or Latino']/df['Total Pop by Hispanic Origin']
/var/folders/g6/p9107ts97cb3b4gk5zy0yfrm0000gn/T/ipykernel_7600/3035970618.py:27: PerformanceWarning: DataFrame is highly fragme
nted.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all column
s at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`
  df['pct renter'] = df['Renter-Occupied Housing']/df['Total by Tenure']
/var/folders/g6/p9107ts97cb3b4gk5zy0yfrm0000gn/T/ipykernel_7600/3035970618.py:28: PerformanceWarning: DataFrame is highly fragme
nted.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all column
s at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`
  df['pct pre-1980 housing'] = (df['Year-built 1970-79']+df['Year-built 1960-69']+df['Year-built 1950-59']+df['Year-built 1940-4
9']+df['Year-built 1939 or earlier'])/df['Total by Year-built']
/var/folders/g6/p9107ts97cb3b4gk5zy0yfrm0000gn/T/ipykernel_7600/3035970618.py:29: PerformanceWarning: DataFrame is highly fragme
nted.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all column
s at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`
  df['pct mobile homes'] = df['Mobile Homes']/df['Total Housing by Type']
/var/folders/g6/p9107ts97cb3b4gk5zy0yfrm0000gn/T/ipykernel_7600/3035970618.py:30: PerformanceWarning: DataFrame is highly fragme
nted.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all column
s at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`
  df['pct no vehicle'] = (df['Owner No Vehicle']+df['Renter No Vehicle'])/df['Total Tenure by Vehicles']
/var/folders/g6/p9107ts97cb3b4gk5zy0yfrm0000gn/T/ipykernel_7600/3035970618.py:31: PerformanceWarning: DataFrame is highly fragme
nted.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all column
s at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`
  df['pct no internet'] = df['No Internet Access']/df['Total Internet Access']
/var/folders/g6/p9107ts97cb3b4gk5zy0yfrm0000gn/T/ipykernel_7600/3035970618.py:33: PerformanceWarning: DataFrame is highly fragme
nted.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all column
s at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`
  df['GEOID_Copy'] = df['GEOID_2']
```

Out[9]:

	NAME	TRACT	Total Pop	Male under 5	Female under 5	Employeed with Disability	Unemployeed with Disability	Not in Labor with Disability	Total Employment by Disability	Agri, Frst, Fish and Mining Workers	Total Workers	Less than High School	Total Education	Income Below Poverty Level	Total Poverty Status
0	Census Tract 4001; Alameda County; California	400100.0	3269.0	72.0	61.0	53.0	9.0	36.0	1724.0	9.0	1641.0	55.0	2522.0	142.0	3248.0
1	Census Tract 4002; Alameda County; California	400200.0	2147.0	83.0	86.0	23.0	0.0	23.0	1244.0	0.0	1124.0	50.0	1740.0	161.0	2147.0
2	Census Tract 4003; Alameda County; California	400300.0	5619.0	34.0	93.0	59.0	96.0	28.0	3761.0	10.0	3350.0	181.0	4514.0	396.0	5592.0

In [10]:
```
# for c in df.columns:
#     print(c)
```

In [11]:
```
## Export the joined dataset to a CSV
df.to_csv('ca_svi_data.csv',index=False)
```

In [12]:
```
## Join this data to the census tract shapefile and export
#read the census tract shapefile
path_ct_geo = "data/tl_2022_06_tract/tl_2022_06_tract.shp"
ca_ct=gpd.read_file(path_ct_geo)
ca_ct.head(2)

#join the svi data to the tracts shapefile
ca_ct['GEOID_2'] = ca_ct['GEOID'].str[1:]
svi_ct = ca_ct.merge(df,how='inner',left_on='GEOID_2',right_on='GEOID_Copy').to_crs(epsg=3857)
svi_ct.head(3)

#export the final shapefile
svi_ct.to_file("ca_svi_ct.shp")
```

```
/var/folders/g6/p9107ts97cb3b4gk5zy0yfrm0000gn/T/ipykernel_7600/1923396632.py:13: UserWarning: Column names longer than 10 characters will be truncated when saved to ESRI Shapefile.
  svi_ct.to_file("ca_svi_ct.shp")
```

OPTIONAL: filter the data to a specific County

In [13]:
```
# Change the county code to a project of need
df = df[df['county'] == 1] #currently this is Alameda County
df.head()
```

Out[13]:

	NAME	TRACT	Total Pop	Male under 5	Female under 5	Employeed with Disability	Unemployeed with Disability	Not in Labor with Disability	Total Employment by Disability	Agri, Frst, Fish and Mining Workers	Total Workers	Less than High School	Total Education	Income Below Poverty Level	Total Poverty Status
0	Census Tract 4001; Alameda County; California	400100.0	3269.0	72.0	61.0	53.0	9.0	36.0	1724.0	9.0	1641.0	55.0	2522.0	142.0	3248.0
1	Census Tract 4002; Alameda County; California	400200.0	2147.0	83.0	86.0	23.0	0.0	23.0	1244.0	0.0	1124.0	50.0	1740.0	161.0	2147.0
2	Census Tract 4003; Alameda County; California	400300.0	5619.0	34.0	93.0	59.0	96.0	28.0	3761.0	10.0	3350.0	181.0	4514.0	396.0	5592.0
3	Census Tract 4004; Alameda County; California	400400.0	4278.0	125.0	197.0	102.0	0.0	166.0	2768.0	11.0	2538.0	78.0	3255.0	419.0	4226.0
4	Census Tract 4005; Alameda County; California	400500.0	3949.0	122.0	36.0	143.0	33.0	19.0	2804.0	0.0	2228.0	65.0	3013.0	409.0	3941.0

2. Exploratory Data Analysis

In [14]:
```
#NaN error handling
print(len(df.index)) #all tracts
print(df.isna().sum().sum()) #all df NaNs

#individual columns' NaN values
print(df['pct hispanic'].isna().sum())
print(df['pct no vehicle'].isna().sum())
print(len(df[df['pct ag workers'].isna()])) #71 NaN values
print(len(df[df['pct ag workers']==0])) #4178 tracts have zero ag workers
```

```
379
52
2
2
2
214
```

In [15]:
```
df_sel = df.iloc[:,-23:] #22 variables in total
df_sel.columns
```

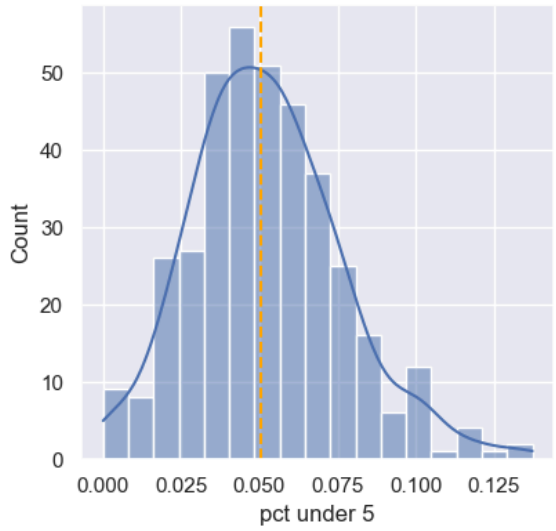
Out[15]:
```
Index(['pct under 5', 'pct disability', 'pct ag workers', 'pct no high school',
       'pct of limited English', 'pct in poverty', 'pct unemployment',
       'pct renter severely burdened', 'pct owner severely burdened',
       'pct non-hispanic white', 'pct non-hispanic black',
       'pct non-hispanic aian', 'pct non-hispanic asian',
       'pct non-hispanic nhpi', 'pct non-hispanic other',
       'pct non-hispanic two', 'pct hispanic', 'pct renter',
       'pct pre-1980 housing', 'pct mobile homes', 'pct no vehicle',
       'pct no internet', 'GEOID_Copy'],
      dtype='object')
```

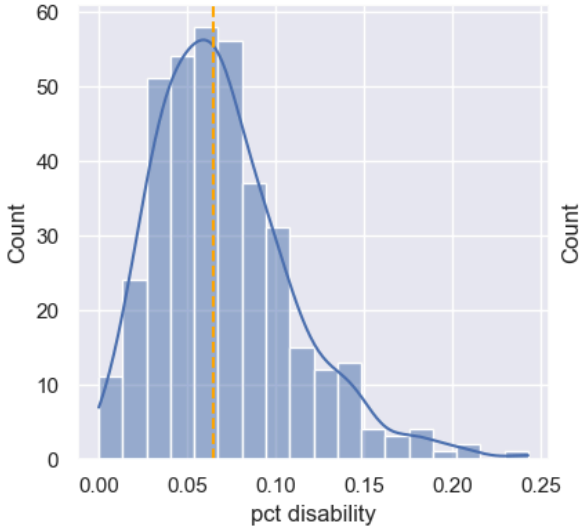
In [16]:
```
# exploratory analysis
df_sel.columns[:-1]
```

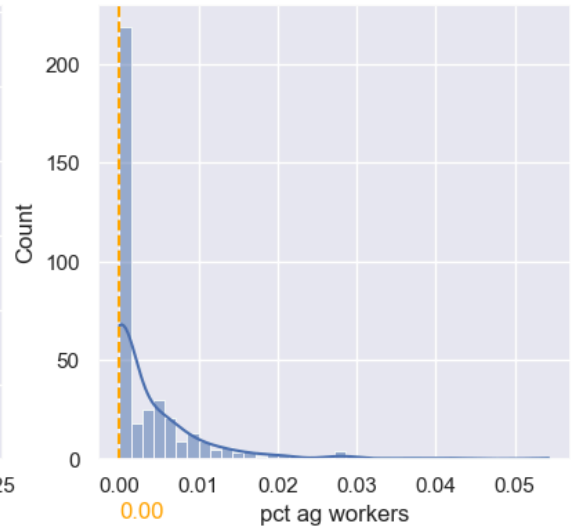
Out[16]:
```
Index(['pct under 5', 'pct disability', 'pct ag workers', 'pct no high school',
       'pct of limited English', 'pct in poverty', 'pct unemployment',
       'pct renter severely burdened', 'pct owner severely burdened',
       'pct non-hispanic white', 'pct non-hispanic black',
       'pct non-hispanic aian', 'pct non-hispanic asian',
       'pct non-hispanic nhpi', 'pct non-hispanic other',
       'pct non-hispanic two', 'pct hispanic', 'pct renter',
       'pct pre-1980 housing', 'pct mobile homes', 'pct no vehicle',
       'pct no internet'],
      dtype='object')
```

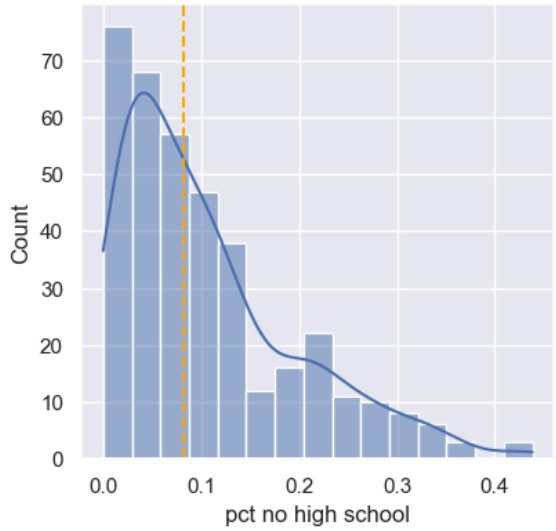
Histograms Analysis

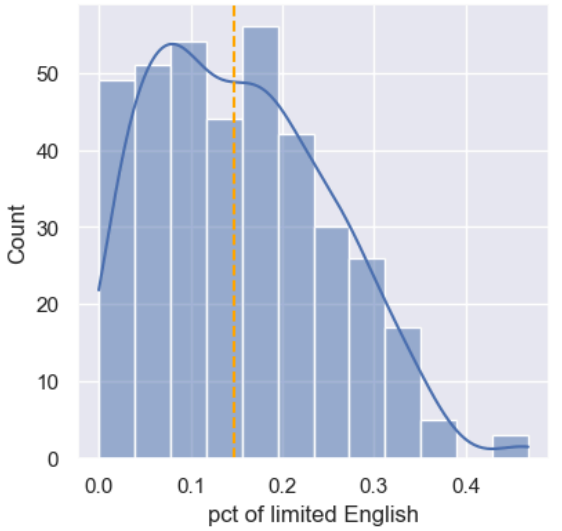
In [17]:
```
# histograms
sb.set(style='darkgrid')
fig,ax = plt.subplots(ncols=3,nrows=7,figsize=(15,35))

r=0
c=0

for r in range(0,7):
    for c in range(0,3):
        i = r*3 + c
        col = df_sel.columns[i]
        sb.histplot(data=df_sel,x=col,kde=True,ax=ax[r,c])
        ax[r,c].axvline(x=df_sel[col].median(), c='orange', ls='--', lw=1.5)
        ax[r,c].text(x=df_sel[col].median(),y=-30,s='{:.2f}'.format(df_sel[col].median()),color='orange')
```

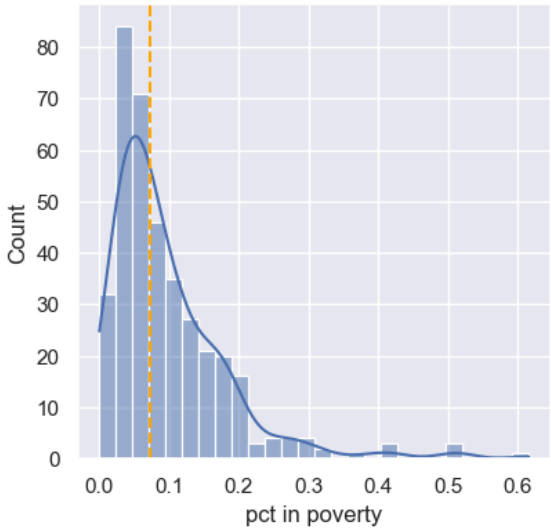

Count
pct under 5

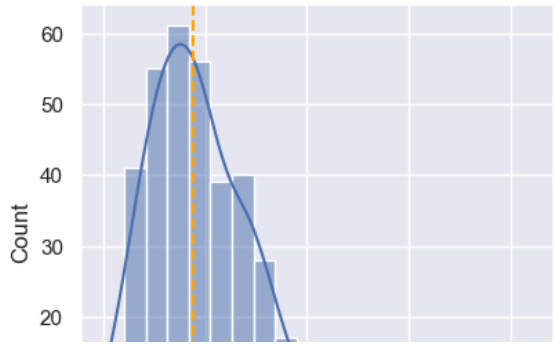

Count
pct disability

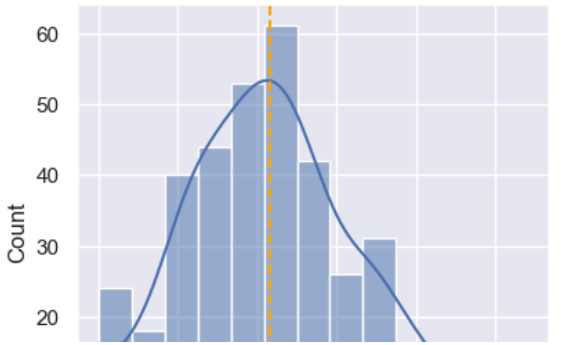

Count
0.00
pct ag workers

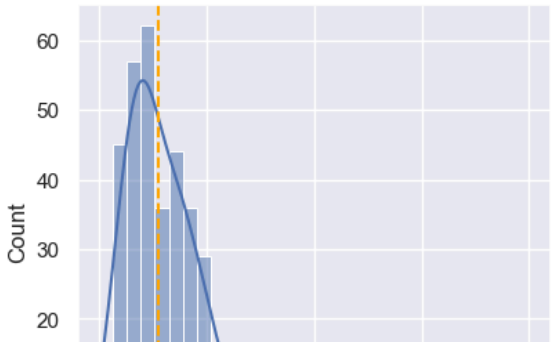

Count
pct no high school

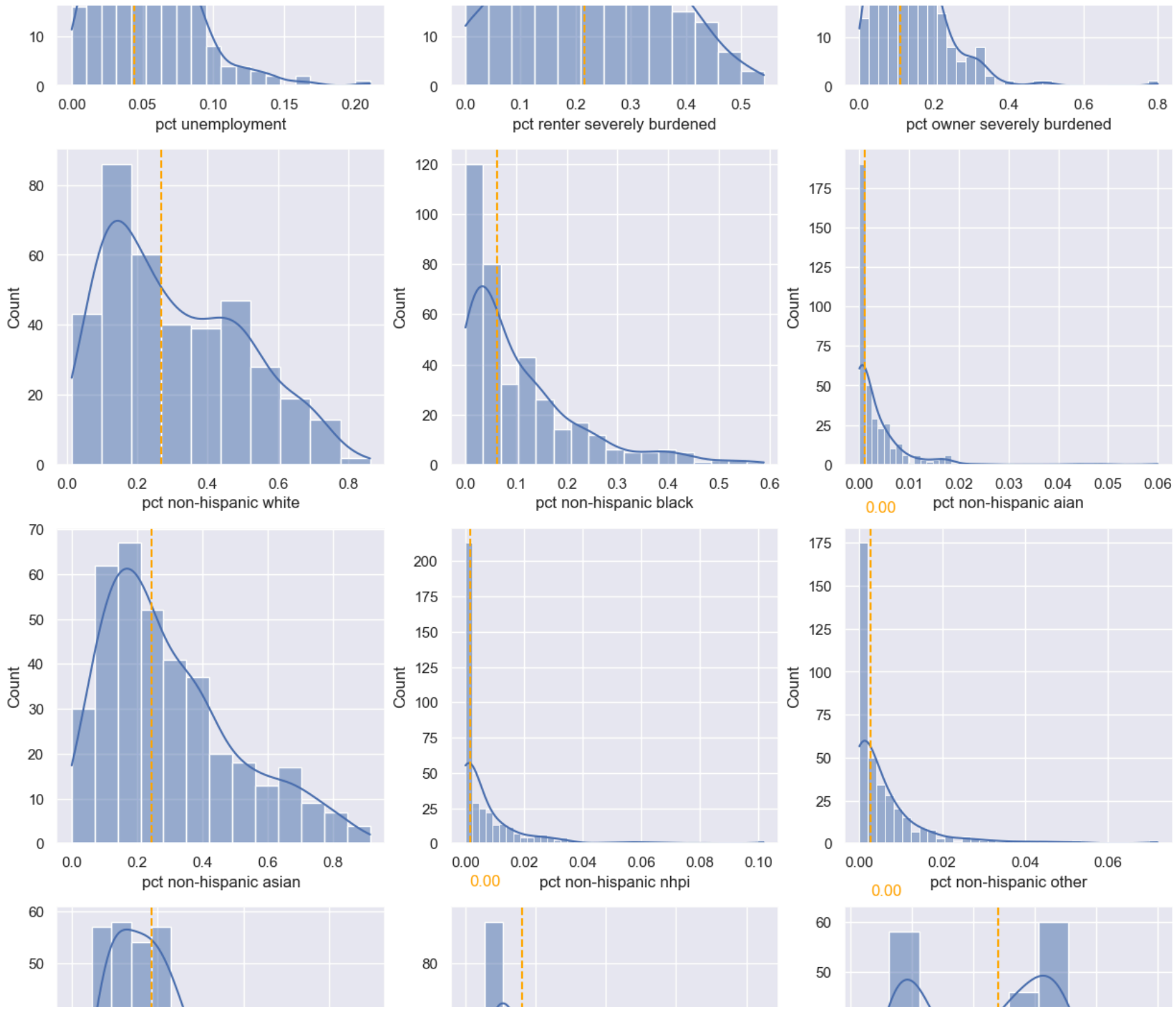

Count
pct of limited English

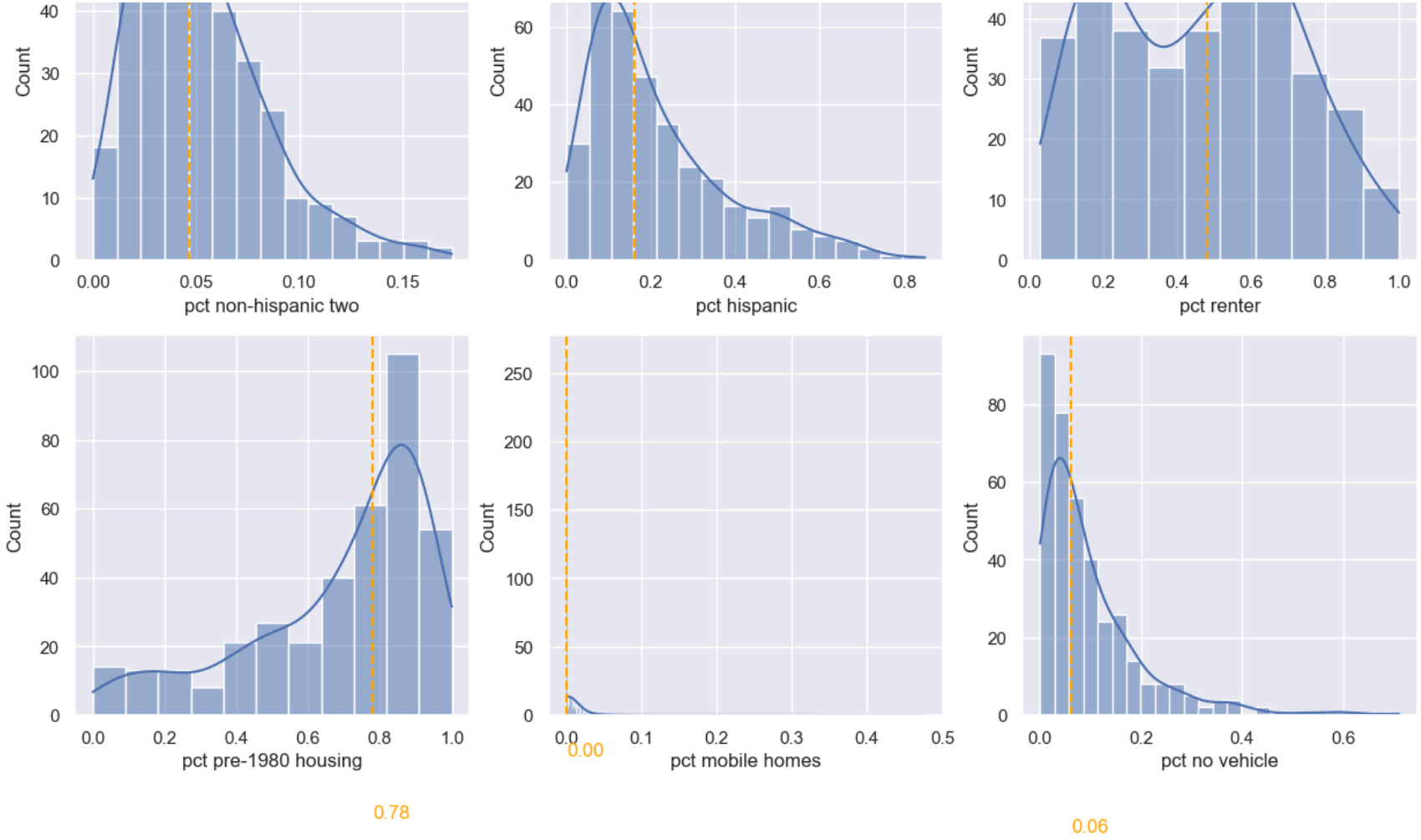

Count
pct in poverty


Count


Count


Count


pct unemployment
pct renter severely burdened
pct owner severely burdened
Count
pct non-hispanic white
pct non-hispanic black
pct non-hispanic aian
0.00
pct non-hispanic asian
pct non-hispanic nhpi
0.00
pct non-hispanic other
0.00



Correlation Analysis

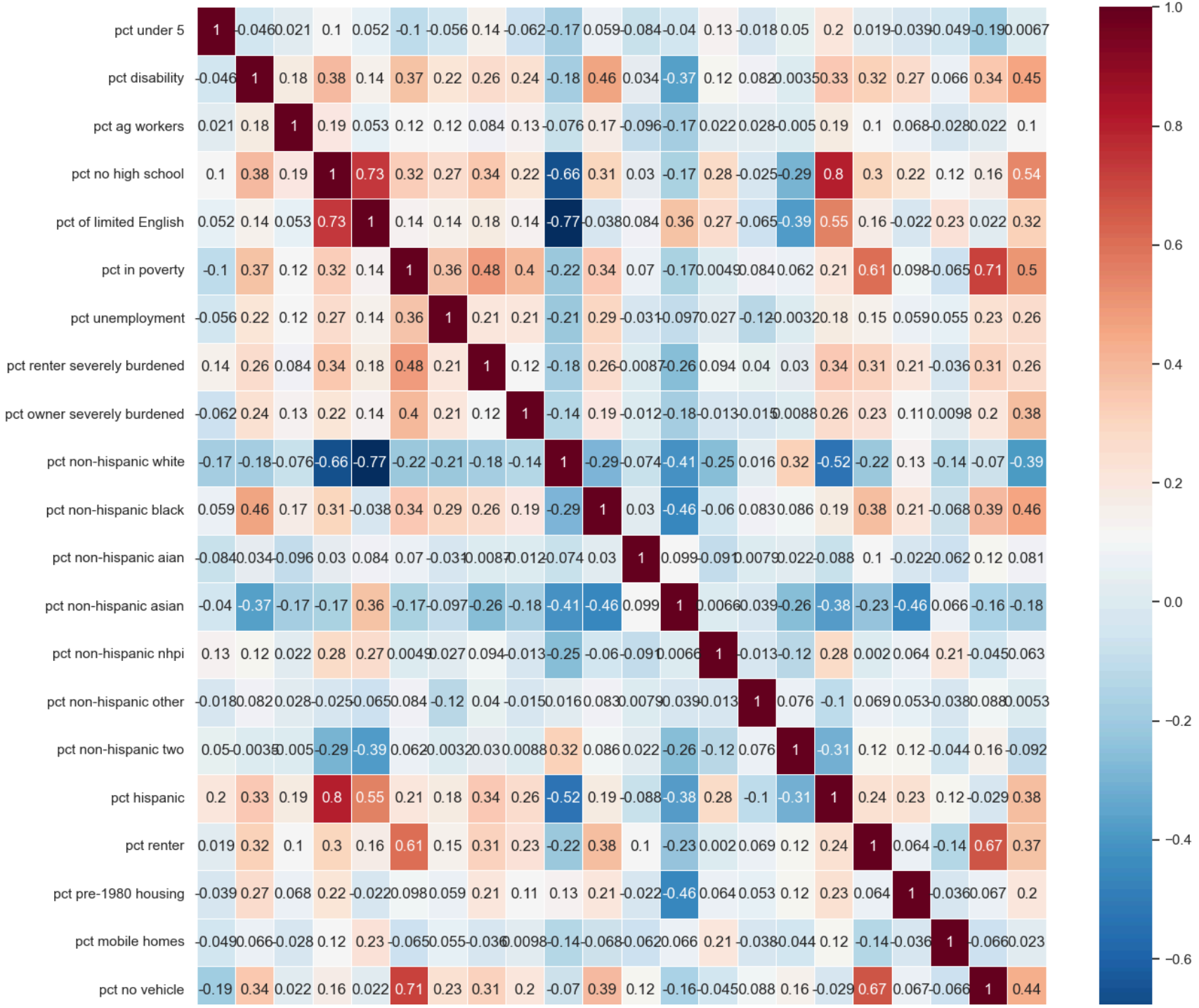
In [18]:
```
# pearson correlation
#ref: using pandas.corr: https://levelup.gitconnected.com/pearson-coefficient-of-correlation-using-pandas-ca68ce678c04
p_corr = df_sel.iloc[:,:-1].corr(method='pearson')
p_corr.head()
```

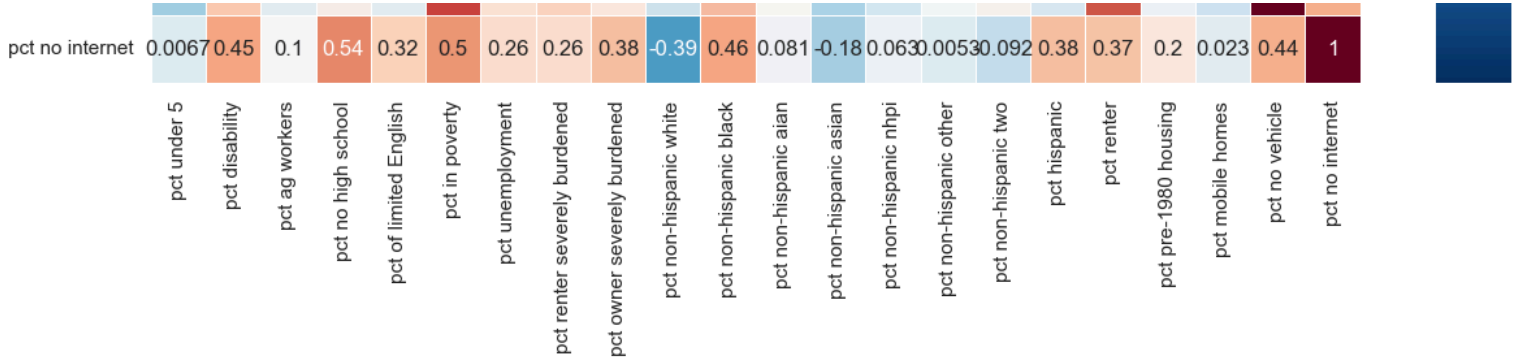
Out[18]:

	pct under 5	pct disability	pct ag workers	pct no high school	pct of limited English	pct in poverty	pct unemployment	pct renter severely burdened	pct owner severely burdened	pct non-hispanic white	pct non-hispanic black	pct non-hispanic aian	pct non-hispanic asian
pct under 5	1.000000	-0.046171	0.021281	0.102061	0.051708	-0.104019	-0.055917	0.141189	-0.061676	-0.173570	0.058597	-0.083844	-0.039626
pct disability	-0.046171	1.000000	0.181525	0.384801	0.141827	0.369239	0.222971	0.256303	0.239619	-0.184361	0.463862	0.033745	-0.370341
pct ag workers	0.021281	0.181525	1.000000	0.193425	0.052753	0.119744	0.115378	0.084159	0.134351	-0.076461	0.166184	-0.096239	-0.172820
pct no high school	0.102061	0.384801	0.193425	1.000000	0.732991	0.322808	0.270608	0.342885	0.218852	-0.656407	0.305508	0.030283	-0.166609
pct of limited English	0.051708	0.141827	0.052753	0.732991	1.000000	0.137794	0.144409	0.176753	0.135190	-0.774023	-0.038023	0.084251	0.363894

In [19]:
```
#plot the correlation
fig,ax = plt.subplots(figsize=(15,15))
sb.heatmap(p_corr,
           xticklabels=p_corr.columns,
           yticklabels=p_corr.columns,
           cmap='RdBu_r',
           annot=True,
           linewidth=0.5,
           ax=ax)
```

Out[19]: <Axes: >


pct under 5 1 -0.046 0.021 0.1 0.052 -0.1 -0.056 0.14 -0.062 -0.17 0.059 -0.084 -0.04 0.13 -0.018 0.05 0.2 0.019 -0.039 0.049 -0.19 0.0067
pct disability -0.046 1 0.18 0.38 0.14 0.37 0.22 0.26 0.24 -0.18 0.46 0.034 -0.37 0.12 0.082 0.0035 0.33 0.32 0.27 0.066 0.34 0.45
pct ag workers 0.021 0.18 1 0.19 0.053 0.12 0.12 0.084 0.13 -0.076 0.17 -0.096 -0.17 0.022 0.028 -0.005 0.19 0.1 0.068 -0.028 0.022 0.1
pct no high school 0.1 0.38 0.19 1 0.73 0.32 0.27 0.34 0.22 -0.66 0.31 0.03 -0.17 0.28 -0.025 -0.29 0.8 0.3 0.22 0.12 0.16 0.54
pct of limited English 0.052 0.14 0.053 0.73 1 0.14 0.14 0.18 0.14 -0.77 -0.038 0.084 0.36 0.27 -0.065 -0.39 0.55 0.16 -0.022 0.23 0.022 0.32
pct in poverty -0.1 0.37 0.12 0.32 0.14 1 0.36 0.48 0.4 -0.22 0.34 0.07 -0.17 0.0049 0.084 0.062 0.21 0.61 0.098 -0.065 0.71 0.5
pct unemployment -0.056 0.22 0.12 0.27 0.14 0.36 1 0.21 0.21 -0.21 0.29 -0.031 -0.097 0.027 -0.12 0.0032 0.18 0.15 0.059 0.055 0.23 0.26
pct renter severely burdened 0.14 0.26 0.084 0.34 0.18 0.48 0.21 1 0.12 -0.18 0.26 -0.0087 -0.26 0.094 0.04 0.03 0.34 0.31 0.21 -0.036 0.31 0.26
pct owner severely burdened -0.062 0.24 0.13 0.22 0.14 0.4 0.21 0.12 1 -0.14 0.19 -0.012 -0.18 -0.013 -0.015 0.0088 0.26 0.23 0.11 0.0098 0.2 0.38
pct non-hispanic white -0.17 -0.18 -0.076 -0.66 -0.77 -0.22 -0.21 -0.18 -0.14 1 -0.29 -0.074 -0.41 -0.25 0.016 0.32 -0.52 -0.22 0.13 -0.14 -0.07 -0.39
pct non-hispanic black 0.059 0.46 0.17 0.31 -0.038 0.34 0.29 0.26 0.19 -0.29 1 0.03 -0.46 -0.06 0.083 0.086 0.19 0.38 0.21 -0.068 0.39 0.46
pct non-hispanic aian -0.084 0.034 -0.096 0.03 0.084 0.07 -0.031 0.0087 0.012 -0.074 0.03 1 0.099 -0.091 0.0079 0.022 -0.088 0.1 -0.022 0.062 0.12 0.081
pct non-hispanic asian -0.04 -0.37 -0.17 -0.17 0.36 -0.17 -0.097 -0.26 -0.18 -0.41 -0.46 0.099 1 0.0066 0.039 -0.26 -0.38 -0.23 -0.46 0.066 -0.16 -0.18
pct non-hispanic nhpi 0.13 0.12 0.022 0.28 0.27 0.0049 0.027 0.094 -0.013 -0.25 -0.06 -0.091 0.0066 1 -0.013 -0.12 0.28 0.002 0.064 0.21 -0.045 0.063
pct non-hispanic other -0.018 0.082 0.028 -0.025 -0.065 0.084 -0.12 0.04 -0.015 0.016 0.083 0.0079 0.039 -0.013 1 0.076 -0.1 0.069 0.053 -0.038 0.088 0.0053
pct non-hispanic two 0.05 -0.0035 0.005 -0.29 -0.39 0.062 0.0032 0.03 0.0088 0.32 0.086 0.022 -0.26 -0.12 0.076 1 -0.31 0.12 0.12 -0.044 0.16 -0.092
pct hispanic 0.2 0.33 0.19 0.8 0.55 0.21 0.18 0.34 0.26 -0.52 0.19 -0.088 -0.38 0.28 -0.1 -0.31 1 0.24 0.23 0.12 -0.029 0.38
pct renter 0.019 0.32 0.1 0.3 0.16 0.61 0.15 0.31 0.23 -0.22 0.38 0.1 -0.23 0.002 0.069 0.12 0.24 1 0.064 -0.14 0.67 0.37
pct pre-1980 housing -0.039 0.27 0.068 0.22 -0.022 0.098 0.059 0.21 0.11 0.13 0.21 -0.022 -0.46 0.064 0.053 0.12 0.23 0.064 1 -0.036 0.067 0.2
pct mobile homes -0.049 0.066 -0.028 0.12 0.23 -0.065 0.055 -0.036 0.0098 -0.14 -0.068 -0.062 0.066 0.21 -0.038 0.044 0.12 -0.14 -0.036 1 -0.066 0.023
pct no vehicle -0.19 0.34 0.022 0.16 0.022 0.71 0.23 0.31 0.2 -0.07 0.39 0.12 -0.16 -0.045 0.088 0.16 -0.029 0.67 0.067 -0.066 1 0.44
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6

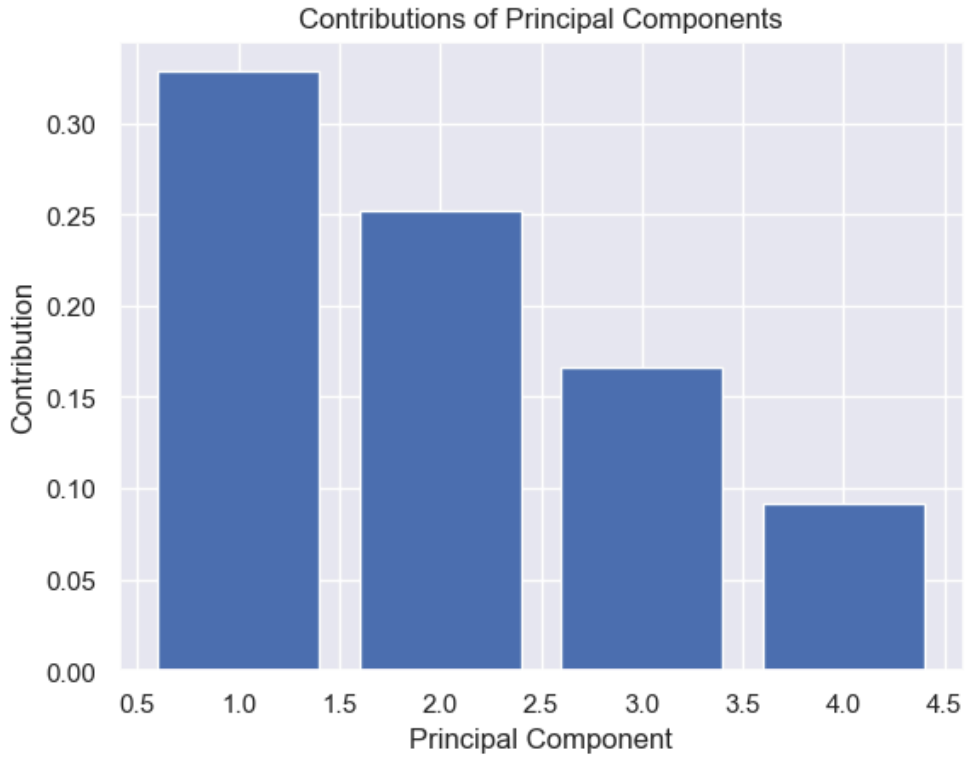


3. Principal Component Analysis (PCA)

In [20]:
```
# PCA requires No NaN values.
# Impute missing values with the median value
imputer = SimpleImputer(strategy='median')
imputed_data = imputer.fit_transform(df_sel.iloc[:,:-1])

# PCA
pca = PCA(n_components=4) # can modify this number depending on specific data
principal_components = pca.fit_transform(imputed_data)


# Create a DataFrame with the principal components
principal_df = pd.DataFrame(data=principal_components, columns=['PC1', 'PC2', 'PC3', 'PC4'])

# Get the explained variance ratio for the PCs
explained_variance_ratio = pca.explained_variance_ratio_

# Print the relative contribution of each PC
print("Contribution by PC:")

for i, ratio in enumerate(explained_variance_ratio):
    print(f"PC{i+1}: {ratio:.2f}")

# Plot the contributions of the PCs
plt.figure()
plt.bar(range(1, len(explained_variance_ratio) + 1), explained_variance_ratio)
# plt.bar_label(label_type='top')
plt.xlabel('Principal Component')
plt.ylabel('Contribution')
plt.title('Contributions of Principal Components')
plt.show()

# Get the variables associated with each PC
```

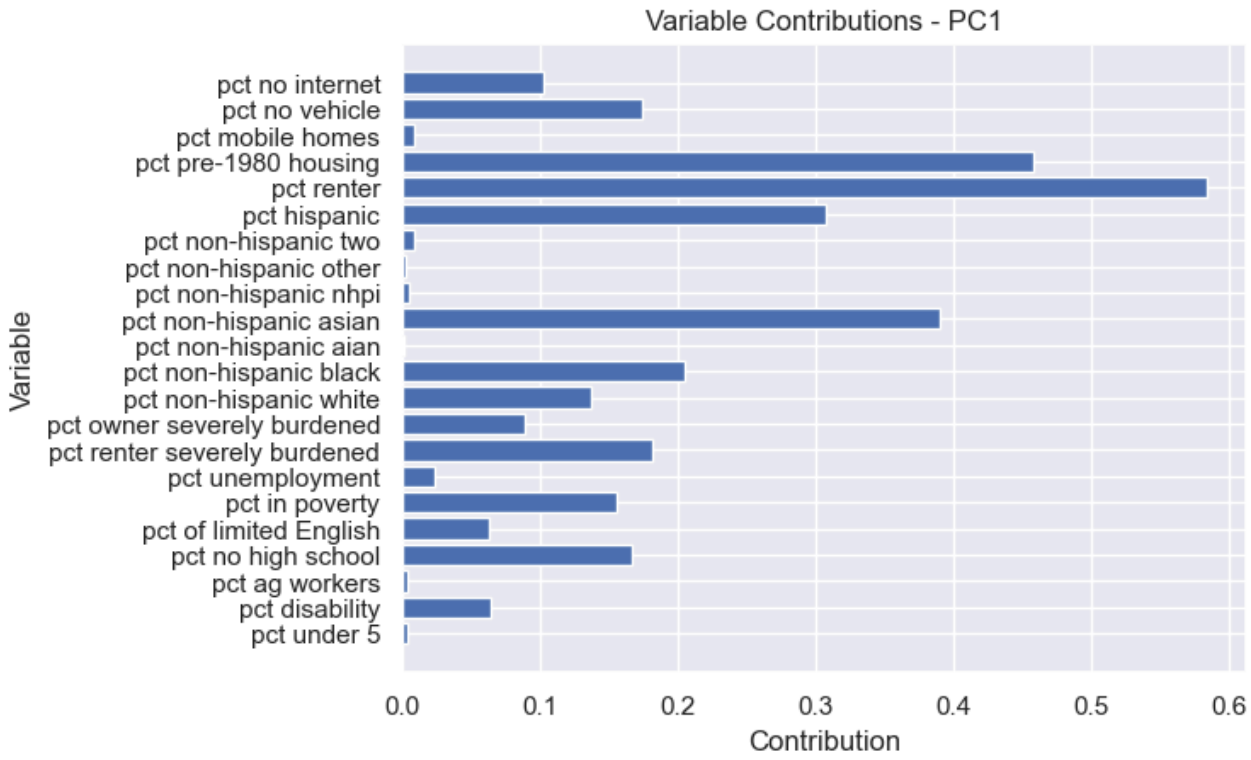
```
component_names = list(df_sel.iloc[:,:-1].columns)
loadings = pca.components_

# Print each variable's contribution to each PC
print("variable contribution for each PC:")

for i, component in enumerate(loadings):
    print(f"PC{i+1}:")
#     for j, variable in enumerate(component_names):
#         contribution = component[j]
#         print(f" - {variable}: {contribution:.2f}")

    # Plot the variable contributions as a bar chart
    magnitudes = np.abs(component)
    # print the exact component names and importance
    print(component_names)
    print(magnitudes)
    df_contributions = pd.concat([pd.Series(component_names), pd.Series(magnitudes)], axis=1)
    print(df_contributions)
    df_contributions.to_csv(f"contributions_PC{i+1}.csv")

    plt.figure()
    plt.barh(component_names, magnitudes)
    plt.xlabel('Contribution')
    plt.ylabel('Variable')
    plt.title(f'Variable Contributions - PC{i+1}')
    plt.show()

    # The magnitude of the contribution, regardless of the sign, indicates the relative importance or weight of the variable in


# Plot the principal components
plt.figure(figsize=(15,15))
plt.scatter(principal_df['PC1'], principal_df['PC2'],s=10)
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('Principal Component Analysis - PC1 & PC2')
plt.show()
```

```
Contribution by PC:
PC1: 0.33
PC2: 0.25
PC3: 0.17
PC4: 0.09
```


Contributions of Principal Components
Contribution
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
Principal Component

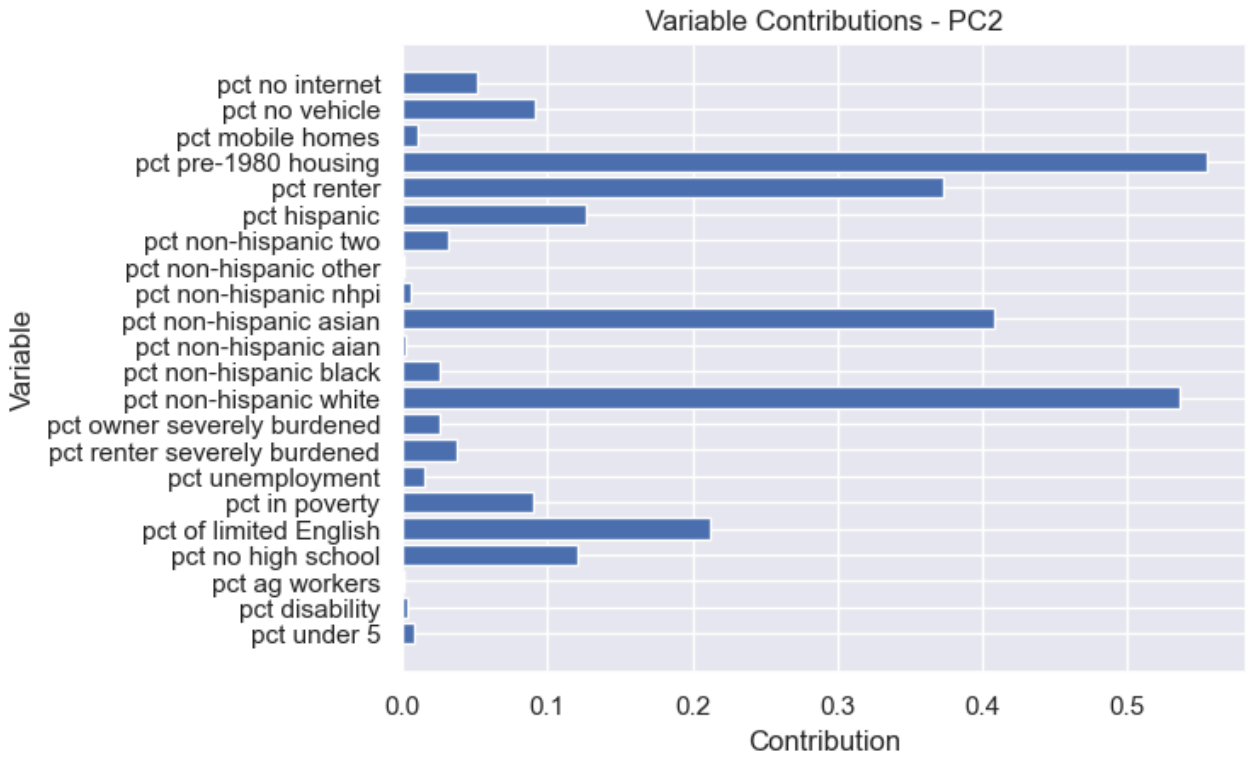
```
variable contribution for each PC:
PC1:
['pct under 5', 'pct disability', 'pct ag workers', 'pct no high school', 'pct of limited English', 'pct in poverty', 'pct unemp
loyment', 'pct renter severely burdened', 'pct owner severely burdened', 'pct non-hispanic white', 'pct non-hispanic black', 'pc
t non-hispanic aian', 'pct non-hispanic asian', 'pct non-hispanic nhpi', 'pct non-hispanic other', 'pct non-hispanic two', 'pct
hispanic', 'pct renter', 'pct pre-1980 housing', 'pct mobile homes', 'pct no vehicle', 'pct no internet']
[3.95012961e-03 6.42907858e-02 3.86251751e-03 1.66166973e-01
 6.30696611e-02 1.55707355e-01 2.33650389e-02 1.81197425e-01
 8.80815782e-02 1.37118351e-01 2.04777706e-01 4.49508878e-04
 3.90463472e-01 4.45333771e-03 1.81851279e-03 8.59541700e-03
 3.07077790e-01 5.83816596e-01 4.58110902e-01 8.95108497e-03
 1.73381521e-01 1.02105801e-01]
```

	0	1
0	pct under 5	0.003950
1	pct disability	0.064291
2	pct ag workers	0.003863
3	pct no high school	0.166167
4	pct of limited English	0.063070
5	pct in poverty	0.155707
6	pct unemployment	0.023365
7	pct renter severely burdened	0.181197
8	pct owner severely burdened	0.088082
9	pct non-hispanic white	0.137118
10	pct non-hispanic black	0.204778
11	pct non-hispanic aian	0.000450
12	pct non-hispanic asian	0.390463
13	pct non-hispanic nhpi	0.004453
14	pct non-hispanic other	0.001819
15	pct non-hispanic two	0.008595
16	pct hispanic	0.307078
17	pct renter	0.583817
18	pct pre-1980 housing	0.458111
19	pct mobile homes	0.008951
20	pct no vehicle	0.173382
21	pct no internet	0.102106

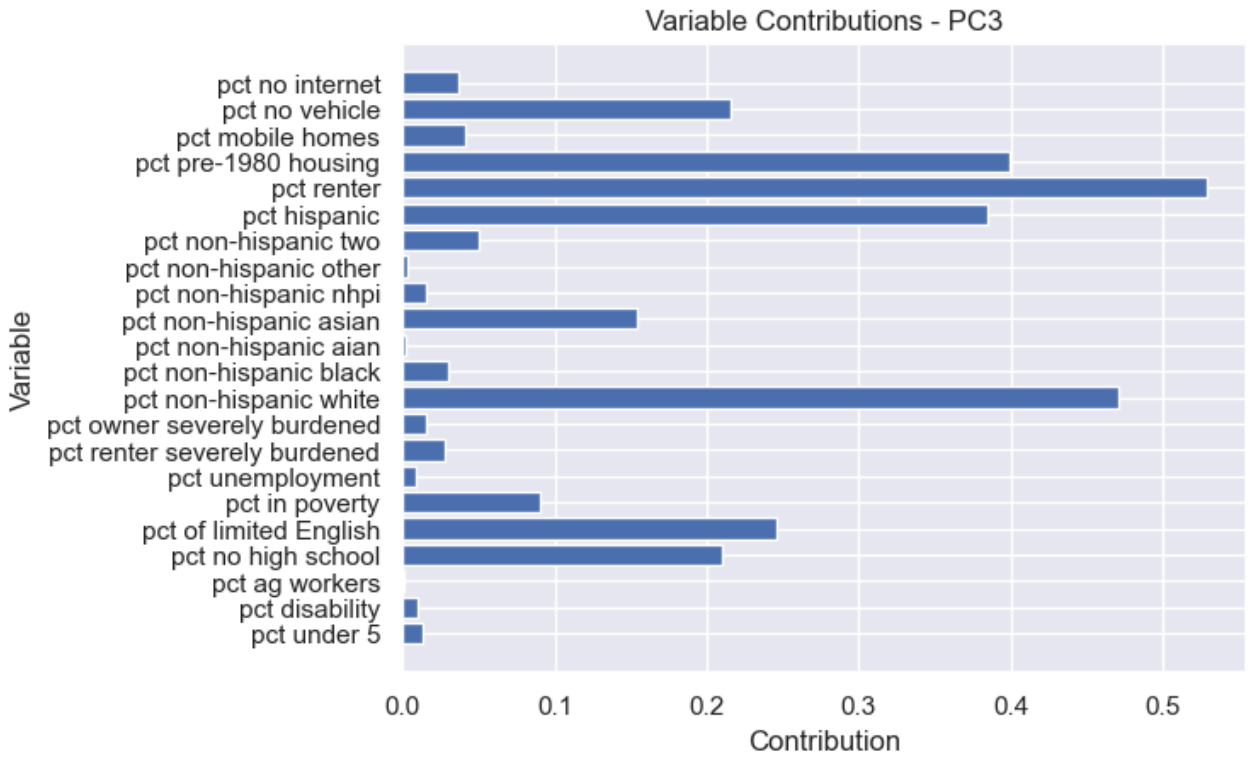

Variable Contributions - PC1
pct no internet
pct no vehicle
pct mobile homes
pct pre-1980 housing
pct renter
pct hispanic
pct non-hispanic two
pct non-hispanic other
pct non-hispanic nhpi
pct non-hispanic asian
pct non-hispanic aian
pct non-hispanic black
pct non-hispanic white
pct owner severely burdened
pct renter severely burdened
pct unemployment
pct in poverty
pct of limited English
pct no high school
pct ag workers
pct disability
pct under 5
Variable
0.0
0.1
0.2
0.3
0.4
0.5
0.6
Contribution

```
PC2:
['pct under 5', 'pct disability', 'pct ag workers', 'pct no high school', 'pct of limited English', 'pct in poverty', 'pct unemp
loyment', 'pct renter severely burdened', 'pct owner severely burdened', 'pct non-hispanic white', 'pct non-hispanic black', 'pc
t non-hispanic aian', 'pct non-hispanic asian', 'pct non-hispanic nhpi', 'pct non-hispanic other', 'pct non-hispanic two', 'pct
hispanic', 'pct renter', 'pct pre-1980 housing', 'pct mobile homes', 'pct no vehicle', 'pct no internet']
[8.03984440e-03 3.06830018e-03 3.21947558e-04 1.20137542e-01
 2.12261421e-01 9.01155620e-02 1.44301023e-02 3.70662239e-02
 2.56330642e-02 5.36234629e-01 2.59211547e-02 2.38301577e-03
 4.07935933e-01 5.63704573e-03 9.99425504e-04 3.08765955e-02
 1.27144410e-01 3.73187997e-01 5.54860630e-01 1.07254218e-02
 9.14408463e-02 5.17021429e-02]
```

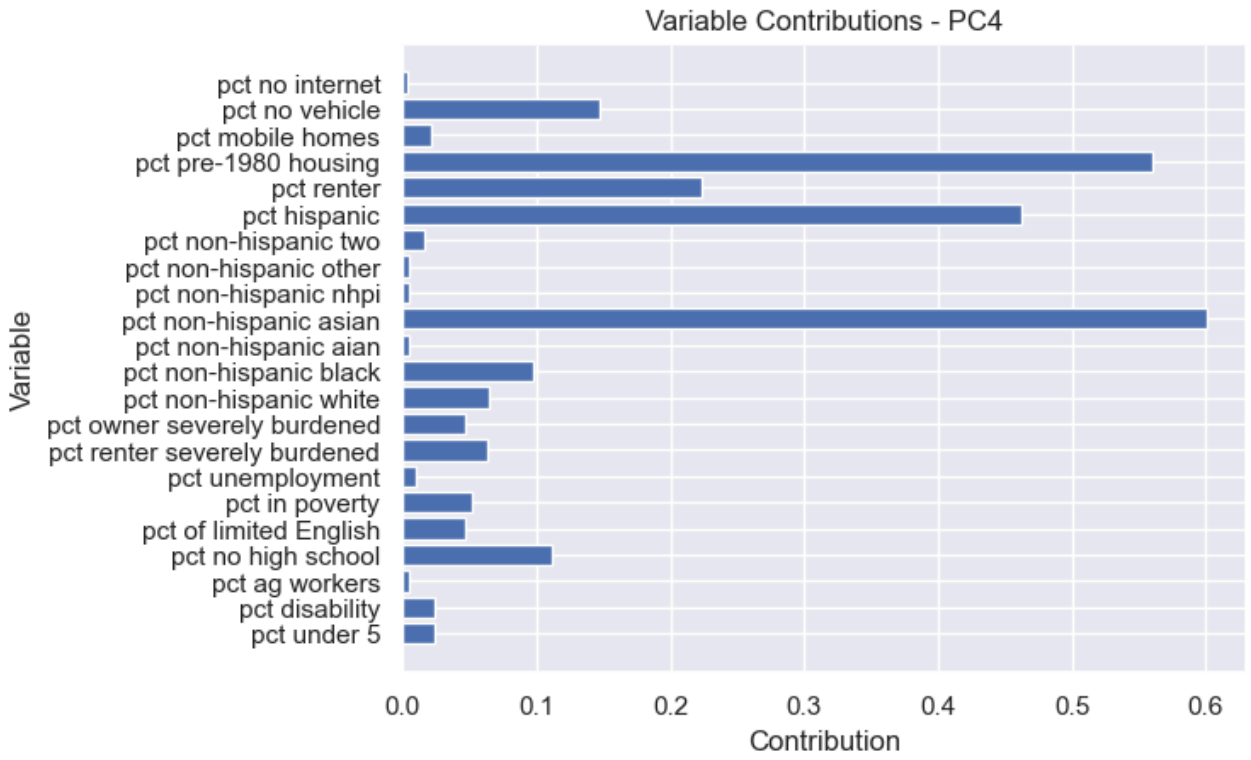
	0	1
0	pct under 5	0.008040
1	pct disability	0.003068
2	pct ag workers	0.000322
3	pct no high school	0.120138
4	pct of limited English	0.212261
5	pct in poverty	0.090116
6	pct unemployment	0.014430
7	pct renter severely burdened	0.037066
8	pct owner severely burdened	0.025633
9	pct non-hispanic white	0.536235
10	pct non-hispanic black	0.025921
11	pct non-hispanic aian	0.002383
12	pct non-hispanic asian	0.407936
13	pct non-hispanic nhpi	0.005637
14	pct non-hispanic other	0.000999
15	pct non-hispanic two	0.030877
16	pct hispanic	0.127144
17	pct renter	0.373188
18	pct pre-1980 housing	0.554861
19	pct mobile homes	0.010725
20	pct no vehicle	0.091441
21	pct no internet	0.051702

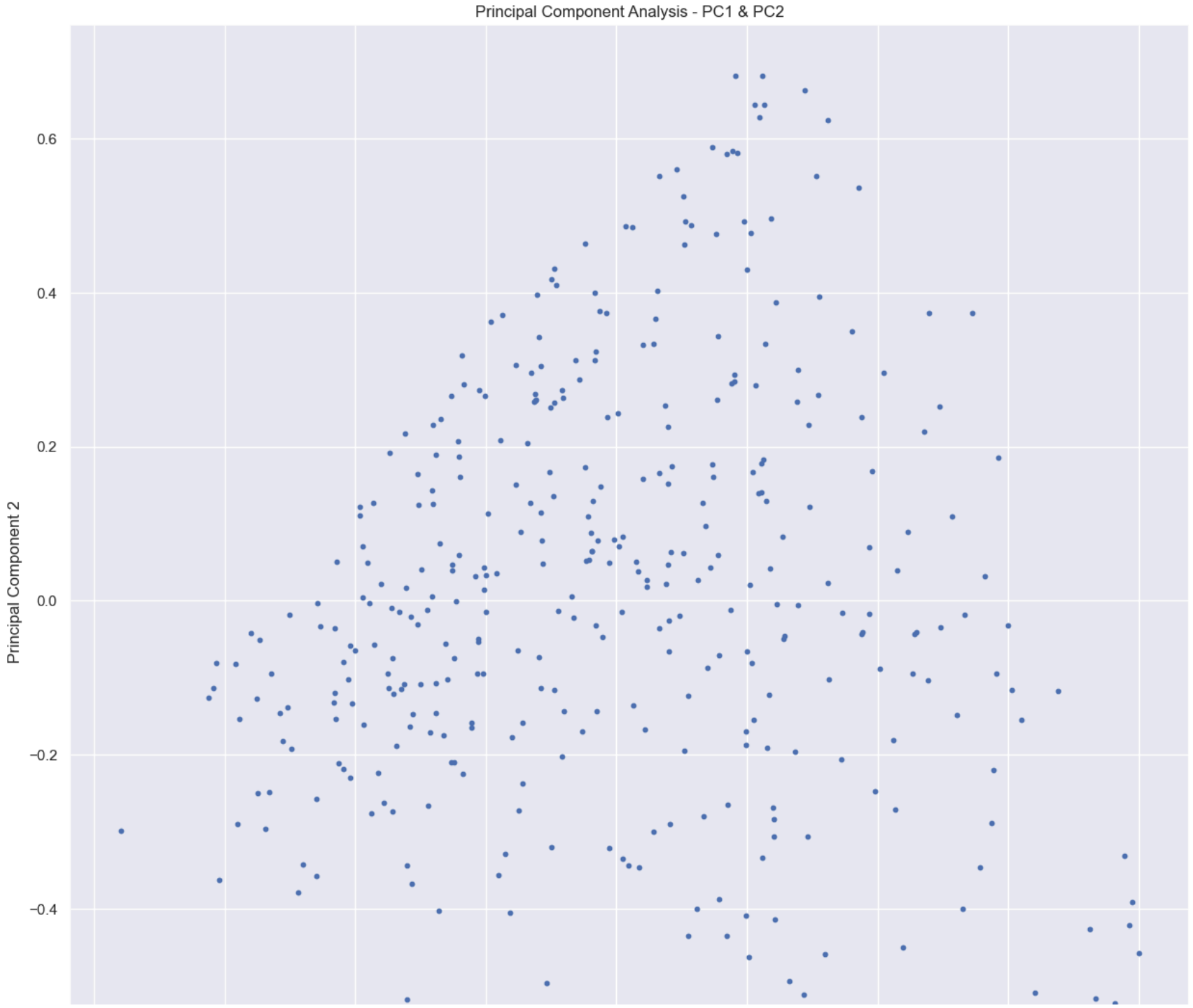

Variable Contributions - PC2
pct no internet
pct no vehicle
pct mobile homes
pct pre-1980 housing
pct renter
pct hispanic
pct non-hispanic two
pct non-hispanic other
pct non-hispanic nhpi
pct non-hispanic asian
pct non-hispanic aian
pct non-hispanic black
pct non-hispanic white
pct owner severely burdened
pct renter severely burdened
pct unemployment
pct in poverty
pct of limited English
pct no high school
pct ag workers
pct disability
pct under 5
Variable
0.0
0.1
0.2
0.3
0.4
0.5
Contribution

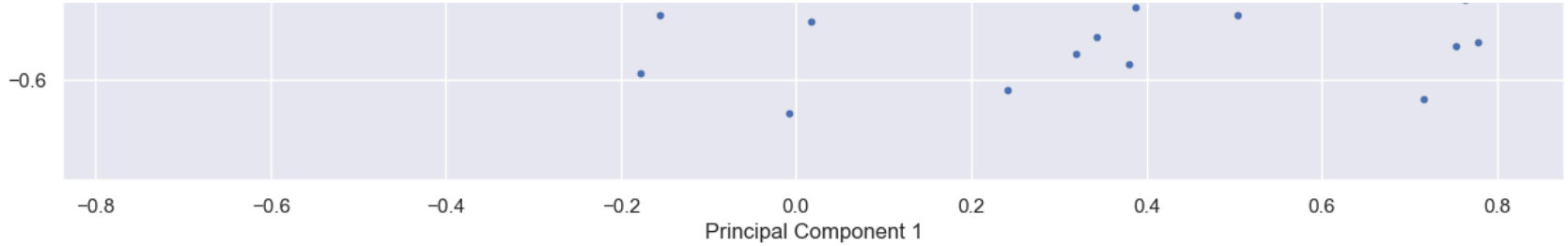
```
PC3:
['pct under 5', 'pct disability', 'pct ag workers', 'pct no high school', 'pct of limited English', 'pct in poverty', 'pct unemp
loyment', 'pct renter severely burdened', 'pct owner severely burdened', 'pct non-hispanic white', 'pct non-hispanic black', 'pc
t non-hispanic aian', 'pct non-hispanic asian', 'pct non-hispanic nhpi', 'pct non-hispanic other', 'pct non-hispanic two', 'pct
hispanic', 'pct renter', 'pct pre-1980 housing', 'pct mobile homes', 'pct no vehicle', 'pct no internet']
[0.01359173 0.00938099 0.00129265 0.21011624 0.24566111 0.0907779
 0.00897804 0.02807832 0.01505468 0.47081428 0.02985652 0.00148426
 0.15448248 0.01584195 0.00361083 0.0501114  0.38509888 0.52845595
 0.3994091  0.04136002 0.21572468 0.03673088]
                               0         1
0                   pct under 5  0.013592
1                 pct disability  0.009381
2                 pct ag workers  0.001293
3             pct no high school  0.210116
4         pct of limited English  0.245661
5                 pct in poverty  0.090778
6               pct unemployment  0.008978
7   pct renter severely burdened  0.028078
8    pct owner severely burdened  0.015055
9         pct non-hispanic white  0.470814
10        pct non-hispanic black  0.029857
11         pct non-hispanic aian  0.001484
12        pct non-hispanic asian  0.154482
13         pct non-hispanic nhpi  0.015842
14        pct non-hispanic other  0.003611
15          pct non-hispanic two  0.050111
16                  pct hispanic  0.385099
17                    pct renter  0.528456
18          pct pre-1980 housing  0.399409
19              pct mobile homes  0.041360
20                pct no vehicle  0.215725
21               pct no internet  0.036731
```


Variable Contributions - PC3
pct no internet
pct no vehicle
pct mobile homes
pct pre-1980 housing
pct renter
pct hispanic
pct non-hispanic two
pct non-hispanic other
pct non-hispanic nhpi
pct non-hispanic asian
pct non-hispanic aian
pct non-hispanic black
pct non-hispanic white
pct owner severely burdened
pct renter severely burdened
pct unemployment
pct in poverty
pct of limited English
pct no high school
pct ag workers
pct disability
pct under 5
Variable
0.0
0.1
0.2
0.3
0.4
0.5
Contribution

```
PC4:
['pct under 5', 'pct disability', 'pct ag workers', 'pct no high school', 'pct of limited English', 'pct in poverty', 'pct unemp
loyment', 'pct renter severely burdened', 'pct owner severely burdened', 'pct non-hispanic white', 'pct non-hispanic black', 'pc
t non-hispanic aian', 'pct non-hispanic asian', 'pct non-hispanic nhpi', 'pct non-hispanic other', 'pct non-hispanic two', 'pct
hispanic', 'pct renter', 'pct pre-1980 housing', 'pct mobile homes', 'pct no vehicle', 'pct no internet']
[0.02341545 0.02444834 0.00522681 0.11203889 0.04650399 0.05206757
 0.009415   0.06326749 0.04701813 0.06434677 0.09705412 0.00514305
 0.60049649 0.00522695 0.00525579 0.01650189 0.46283122 0.22341936
 0.56071927 0.02152826 0.14689188 0.0035631 ]
                                0         1
0                   pct under 5  0.023415
1                 pct disability  0.024448
2                 pct ag workers  0.005227
3             pct no high school  0.112039
4         pct of limited English  0.046504
5                 pct in poverty  0.052068
6               pct unemployment  0.009415
7   pct renter severely burdened  0.063267
8    pct owner severely burdened  0.047018
9         pct non-hispanic white  0.064347
10        pct non-hispanic black  0.097054
11         pct non-hispanic aian  0.005143
12        pct non-hispanic asian  0.600496
13         pct non-hispanic nhpi  0.005227
14        pct non-hispanic other  0.005256
15          pct non-hispanic two  0.016502
16                  pct hispanic  0.462831
17                    pct renter  0.223419
18          pct pre-1980 housing  0.560719
19              pct mobile homes  0.021528
20                pct no vehicle  0.146892
21               pct no internet  0.003563
```


Variable Contributions - PC4
pct no internet
pct no vehicle
pct mobile homes
pct pre-1980 housing
pct renter
pct hispanic
pct non-hispanic two
pct non-hispanic other
pct non-hispanic nhpi
pct non-hispanic asian
pct non-hispanic aian
pct non-hispanic black
pct non-hispanic white
pct owner severely burdened
pct renter severely burdened
pct unemployment
pct in poverty
pct of limited English
pct no high school
pct ag workers
pct disability
pct under 5
Variable
0.0
0.1
0.2
0.3
0.4
0.5
0.6
Contribution


Principal Component Analysis - PC1 & PC2
Principal Component 2
0.6
0.4
0.2
0.0
−0.2
−0.4



Optional TBD: variable selection based on PCA

In [21]:
```
df_sel.columns
```

Out[21]:
```
Index(['pct under 5', 'pct disability', 'pct ag workers', 'pct no high school',
       'pct of limited English', 'pct in poverty', 'pct unemployment',
       'pct renter severely burdened', 'pct owner severely burdened',
       'pct non-hispanic white', 'pct non-hispanic black',
       'pct non-hispanic aian', 'pct non-hispanic asian',
       'pct non-hispanic nhpi', 'pct non-hispanic other',
       'pct non-hispanic two', 'pct hispanic', 'pct renter',
       'pct pre-1980 housing', 'pct mobile homes', 'pct no vehicle',
       'pct no internet', 'GEOID_Copy'],
      dtype='object')
```

In [22]:
```
df_sel.head()
df_sel.dtypes
```

Out[22]:
```
pct under 5                    float64
pct disability                 float64
pct ag workers                 float64
pct no high school             float64
pct of limited English         float64
pct in poverty                 float64
pct unemployment               float64
pct renter severely burdened   float64
pct owner severely burdened    float64
pct non-hispanic white         float64
pct non-hispanic black         float64
pct non-hispanic aian          float64
pct non-hispanic asian         float64
pct non-hispanic nhpi          float64
pct non-hispanic other         float64
pct non-hispanic two           float64
pct hispanic                   float64
pct renter                     float64
pct pre-1980 housing           float64
pct mobile homes               float64
pct no vehicle                 float64
pct no internet                float64
GEOID_Copy                      object
dtype: object
```

4. Optional: Machine Learning Predicitve Data Analysis (using SB535 DAC as proxy)

In [23]:
```
# use this col to join with the SB535 DAC layer - df_sel['GEOID_Copy']
#read the preprocessed sb535 dac pt shapefile
path_sb535_dac_pt = "data/pt_ct_SB535_DAC_joined.shp"
sb535_dac_pt=gpd.read_file(path_sb535_dac_pt)
sb535_dac_pt['GEOID'] = sb535_dac_pt['GEOID'].astype(str).str[1:]
sb535_dac_pt.head(2)
```

Out[23]:

	GEOID	NAME	NAMELSAD	SB535_DAC	GEOID_N	geometry
0	6065940600	9406	Census Tract 9406	N	6065940600	POINT (-116.42333 33.82323)
1	6065940700	9407	Census Tract 9407	N	6065940700	POINT (-116.51946 33.80845)

In [24]:
```
df_sel_dac = df_sel.merge(sb535_dac_pt,how='inner',left_on='GEOID_Copy',right_on='GEOID')
df_sel_dac = df_sel_dac.drop(columns=['GEOID','NAME','NAMELSAD','GEOID_N','geometry'])
df_sel_dac.head()
```

Out[24]:

	pct under 5	pct disability	pct ag workers	pct no high school	pct of limited English	pct in poverty	pct unemployment	pct renter severely burdened	pct owner severely burdened	pct non-hispanic white	pct non-hispanic black	pct non-hispanic aian	pct non-hispanic asian	pct non-hispanic nhpi
0	0.040685	0.056845	0.005484	0.021808	0.015944	0.043719	0.053584	0.000000	0.203200	0.689813	0.052310	0.000000	0.145916	0.000000
1	0.078714	0.036977	0.000000	0.028736	0.019717	0.074988	0.067186	0.072539	0.102041	0.698649	0.017699	0.004192	0.116907	0.000000
2	0.022602	0.048657	0.002985	0.040097	0.022396	0.070815	0.077107	0.111768	0.055966	0.585513	0.094145	0.004627	0.113009	0.000000
3	0.075269	0.096821	0.004334	0.023963	0.014914	0.099148	0.000000	0.192785	0.100802	0.640486	0.116643	0.005376	0.087190	0.000000
4	0.040010	0.069544	0.000000	0.021573	0.027961	0.103781	0.081315	0.256264	0.300874	0.414789	0.251456	0.001013	0.093695	0.001519

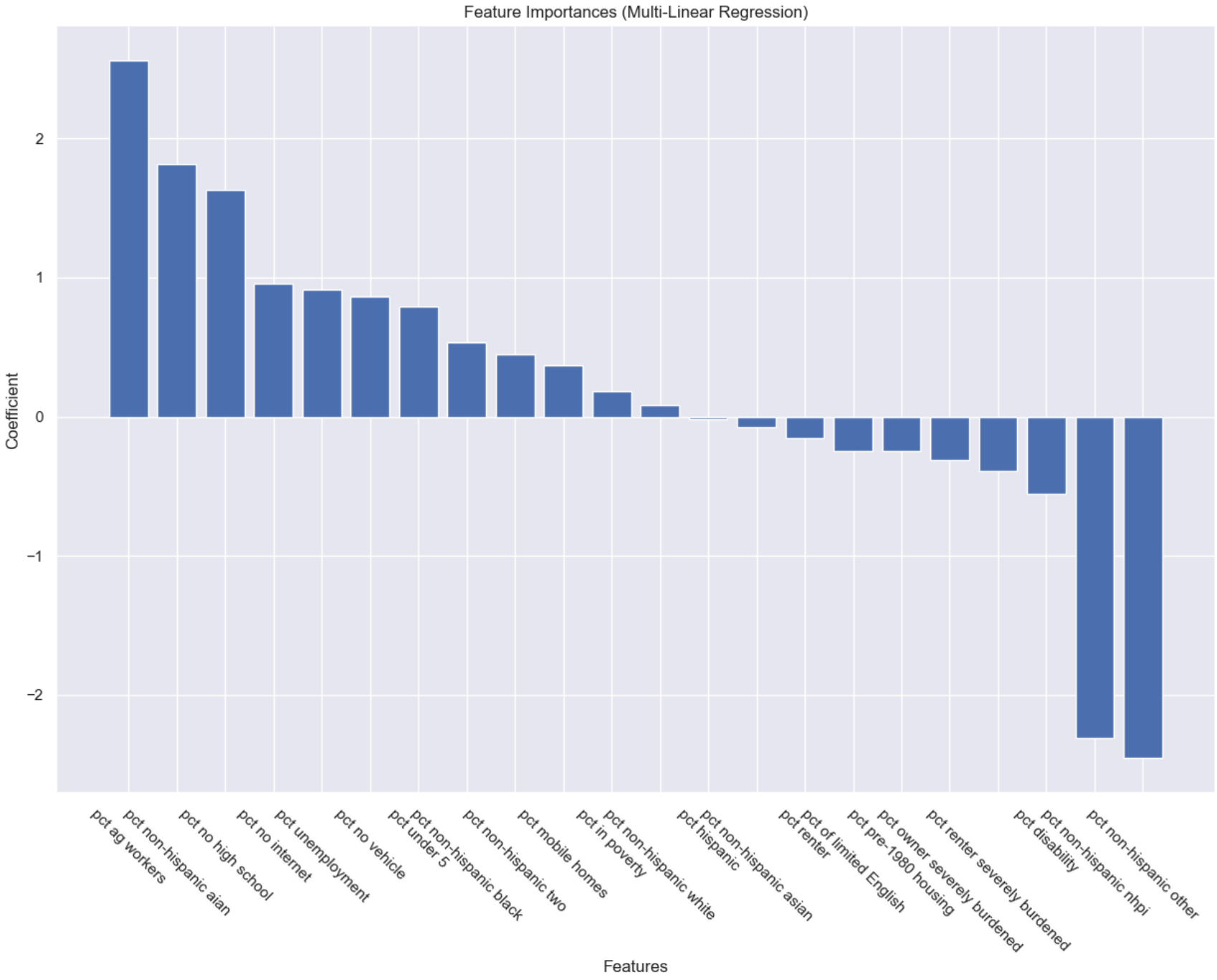
Linear Regression

In [25]:

```
# Linear Regression on SB535 DAC with Accuracy
# MODEL 1: apply Logistic Regression on SB535 disadvantaged communities. Note that Logistic Regression does not allow NaN values
X = df_sel_dac.iloc[:,:-2]
y = df_sel_dac.iloc[:,-1]
y_binary = np.where(y == 'Y', 1, 0) #assign DAC as 1, otherwise 0
y_binary
# np.where(y_binary == 1)

#Impute missing values with the median value
imputer = SimpleImputer(strategy='median')
X_imputed = imputer.fit_transform(X)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_imputed, y_binary, test_size=0.2, random_state=42) #random_state=42 is the

# Create and fit the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Print the accuracy level
accuracy = model.score(X_test, y_test)
print("R-squared:", str((accuracy*100).round(2)) + "%")

# Get the coefficients (feature importances) of the model
coefficients = model.coef_
feature_names = X.columns.tolist()

# Create a DataFrame to store the results
importance_df = pd.DataFrame({'Feature': feature_names, 'Coefficient': coefficients})

# Sort the DataFrame by coefficient values
importance_df = importance_df.sort_values(by='Coefficient', ascending=False).reset_index(drop=True)
```

```
# Plot the feature importances
plt.figure(figsize=(15, 10))
plt.bar(importance_df['Feature'], importance_df['Coefficient'])
plt.xlabel('Features')
plt.ylabel('Coefficient')
plt.title('Feature Importances (Multi-Linear Regression)')
plt.xticks(rotation=-45)
# plt.tight_layout()
plt.show()
```

```
R-squared: -5.98%
```

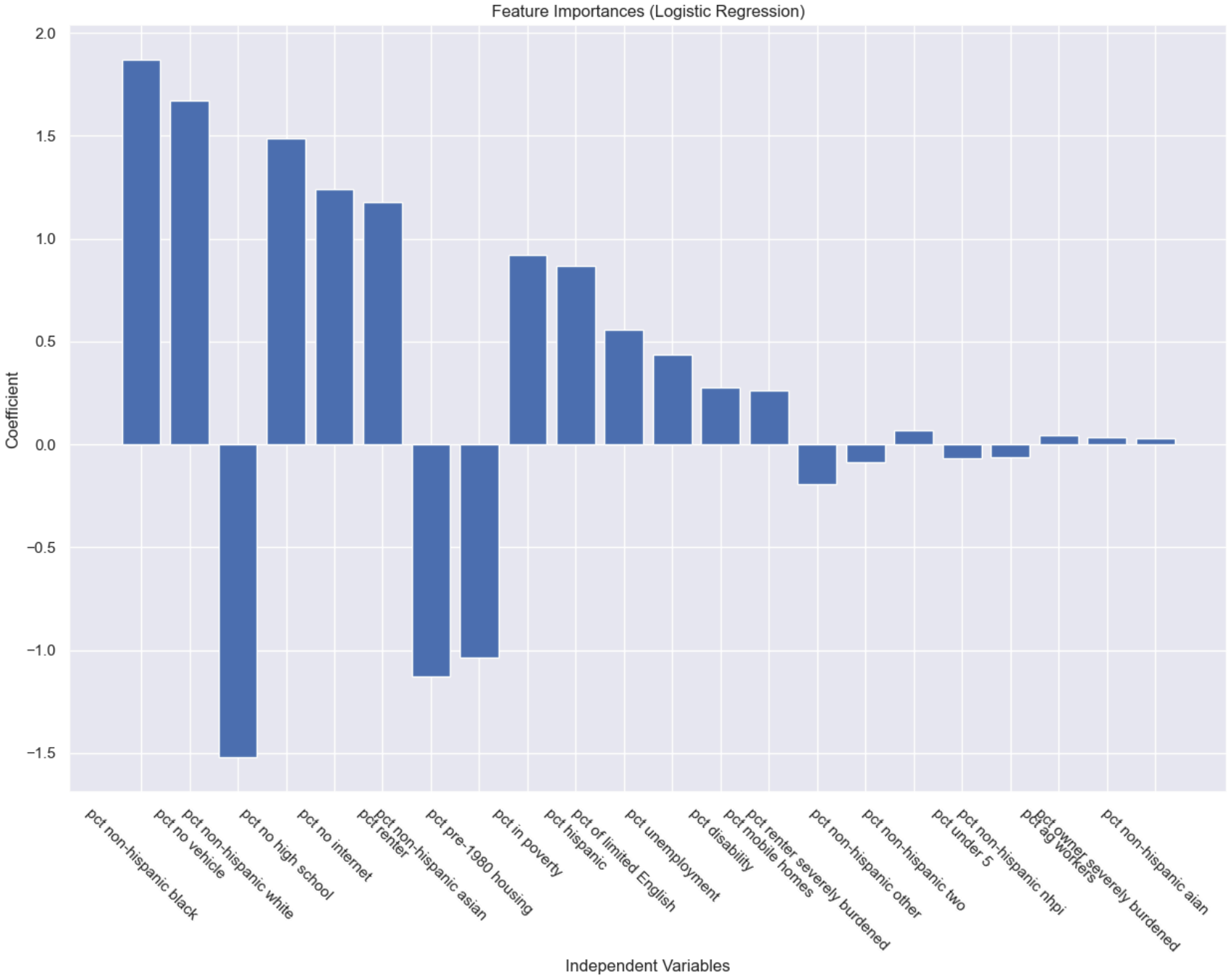

Feature Importances (Multi-Linear Regression)
Coefficient
2
1
0
−1
−2
pct ag workers
pct non-hispanic aian
pct no high school
pct no internet
pct unemployment
pct no vehicle
pct under 5
pct non-hispanic black
pct non-hispanic two
pct mobile homes
pct in poverty
pct non-hispanic white
pct hispanic
pct non-hispanic asian
pct renter
pct of limited English
pct pre-1980 housing
pct owner severely burdened
pct renter severely burdened
pct disability
pct non-hispanic nhpi
pct non-hispanic other
Features

Machine Learning

In [26]:
```
### MACHINE LEARNING (Classification) Models ###

# MODEL 1: apply Logistic Regression on SB535 disadvantaged communities. Note that Logistic Regression does not allow NaN values
X = df_sel_dac.iloc[:,:-2]
y = df_sel_dac.iloc[:,-1]
y_binary = np.where(y == 'Y', 1, 0) #assign DAC as 1, otherwise 0
y_binary
# np.where(y_binary == 1)

#Impute missing values with the median value
imputer = SimpleImputer(strategy='median')
X_imputed = imputer.fit_transform(X)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_imputed, y_binary, test_size=0.2, random_state=42) #random_state=42 is the

# Create and fit the logistic regression model
model = LogisticRegression()
model.fit(X_train, y_train)

# Print the accuracy level
accuracy = model.score(X_test, y_test)
print("Model Accuracy:", str((accuracy*100).round(2)) + "%")

# Get the coefficient values and corresponding feature names
coefs = model.coef_[0]
feature_names = X.columns.tolist()

# Create a DataFrame to display the results
importance_df = pd.DataFrame({'Feature': feature_names, 'Coefficient': coefs})

# Sort the DataFrame by absolute coefficient values
importance_df['Absolute Coefficient'] = importance_df['Coefficient'].abs()
importance_df = importance_df.sort_values(by='Absolute Coefficient', ascending=False).reset_index(drop=True)

# Print the relative importance of each independent variable
print(importance_df)

# plot it
plt.figure(figsize=(15, 10))
plt.bar(importance_df['Feature'], importance_df['Coefficient'])
plt.xlabel('Independent Variables')
plt.ylabel('Coefficient')
plt.title('Feature Importances (Logistic Regression)')
plt.xticks(rotation=-45)
```

```
# plt.tight_layout()
plt.show()
```

```
Model Accuracy: 88.16%
                         Feature  Coefficient  Absolute Coefficient
0         pct non-hispanic black     1.868325              1.868325
1                 pct no vehicle     1.669044              1.669044
2         pct non-hispanic white    -1.521978              1.521978
3             pct no high school     1.488340              1.488340
4                pct no internet     1.238433              1.238433
5                     pct renter     1.177232              1.177232
6         pct non-hispanic asian    -1.125672              1.125672
7           pct pre-1980 housing    -1.036063              1.036063
8                 pct in poverty     0.922378              0.922378
9                   pct hispanic     0.865547              0.865547
10        pct of limited English     0.559477              0.559477
11              pct unemployment     0.435700              0.435700
12                pct disability     0.275748              0.275748
13              pct mobile homes     0.263155              0.263155
14  pct renter severely burdened    -0.194043              0.194043
15        pct non-hispanic other    -0.087976              0.087976
16          pct non-hispanic two     0.069616              0.069616
17                   pct under 5    -0.066675              0.066675
18         pct non-hispanic nhpi    -0.064278              0.064278
19                pct ag workers     0.044119              0.044119
20   pct owner severely burdened     0.032776              0.032776
21         pct non-hispanic aian     0.029063              0.029063
```


Feature Importances (Logistic Regression)
Coefficient
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
pct non-hispanic black
pct no vehicle
pct non-hispanic white
pct no high school
pct no internet
pct renter
pct non-hispanic asian
pct pre-1980 housing
pct in poverty
pct hispanic
pct of limited English
pct unemployment
pct disability
pct mobile homes
pct renter severely burdened
pct non-hispanic other
pct non-hispanic two
pct under 5
pct non-hispanic nhpi
pct ag workers
pct owner severely burdened
pct non-hispanic aian
Independent Variables

In [27]:
```
# Using simplified data from PCA - accuracy dropped a bit
# MODEL 1 with PCA: apply Logistic Regression on SB535 disadvantaged communities. Note that Logistic Regression does not allow N
X = df_sel_dac.iloc[:,:-2]
y = df_sel_dac.iloc[:,-1]
y_binary = np.where(y == 'Y', 1, 0) #assign DAC as 1, otherwise 0
y_binary

#Impute missing values with the median value
imputer = SimpleImputer(strategy='median')
X_imputed = imputer.fit_transform(X)

# Perform PCA for dimensionality reduction
pca = PCA(n_components=0.95)  # Specify the desired % of variance preservation
X_pca = pca.fit_transform(X_imputed)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_pca, y_binary, test_size=0.2, random_state=42) #random_state=42 is the see

# Create and fit the logistic regression model
model = LogisticRegression()
model.fit(X_train, y_train)

# Print the accuracy level
accuracy = model.score(X_test, y_test)
print("Model Accuracy:", str((accuracy*100).round(2)) + "%")

## Following will have errors with PCA transformed data!!!
# # Get the coefficient values and corresponding feature names
# coefs = model.coef_[0]
# feature_names = X.columns.tolist()

# # Create a DataFrame to display the results
# importance_df = pd.DataFrame({'Feature': feature_names, 'Coefficient': coefs})

# # Sort the DataFrame by absolute coefficient values
# importance_df['Absolute Coefficient'] = importance_df['Coefficient'].abs()
# importance_df = importance_df.sort_values(by='Absolute Coefficient', ascending=False).reset_index(drop=True)

# # Print the relative importance of each independent variable
# print(importance_df)

# # plot it
# plt.figure(figsize=(15, 10))
# plt.bar(importance_df['Feature'], importance_df['Coefficient'])
# plt.xlabel('Independent Variables')
# plt.ylabel('Coefficient')
# plt.title('Feature Importances (Logistic Regression)')
# plt.xticks(rotation=-45)
```

```
# # plt.tight_layout()
# plt.show()
```

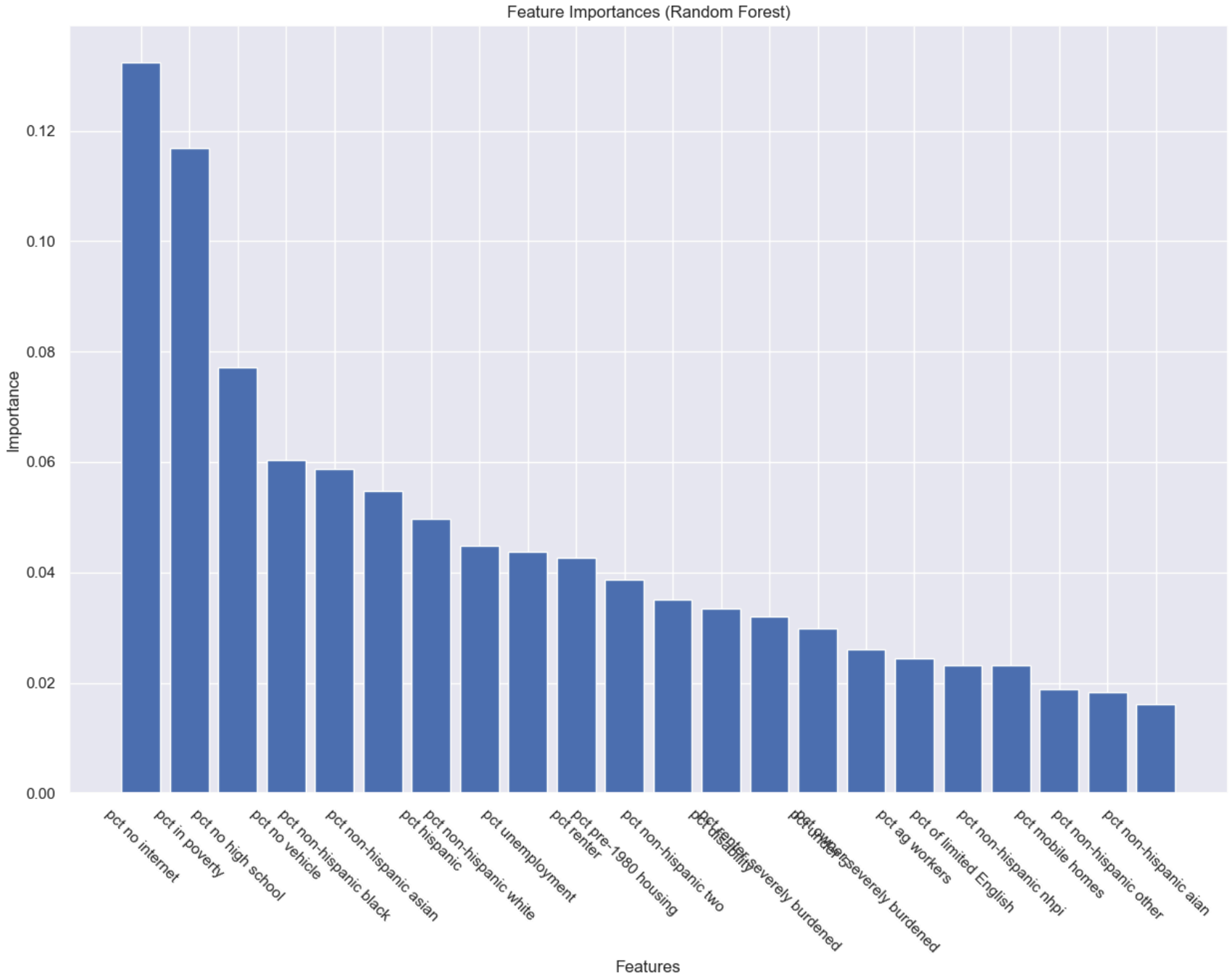
```
Model Accuracy: 88.16%
```

In [75]:
```
### Other ML Models Selection

# There are several alternative machine learning models that you can consider for a classification problem like logistic regress

# Random Forest: Random Forest is an ensemble learning method that combines multiple decision trees. It can handle both numerica

# Support Vector Machines (SVM): SVM is a powerful algorithm for classification tasks. It aims to find the optimal hyperplane th

# Gradient Boosting: Gradient Boosting algorithms, such as XGBoost or LightGBM, build an ensemble of weak prediction models in a

# Neural Networks: Deep learning models, particularly neural networks, have gained popularity in recent years. They are capable

# Naive Bayes: Naive Bayes is a probabilistic classifier based on Bayes' theorem. It assumes that features are independent of ea

# Decision Trees: Decision Trees are simple yet effective models for classification problems. They split the data based on diffe
```

In [28]:
```
### MACHINE LEARNING (Classification) Models ###

# MODEL 2: Random Forest Model
X = df_sel_dac.iloc[:,:-2]
y = df_sel_dac.iloc[:,-1]
y_binary = np.where(y == 'Y', 1, 0) #assign DAC as 1, otherwise 0
y_binary
# np.where(y_binary == 1)

#Impute missing values with the median value
imputer = SimpleImputer(strategy='median')
X_imputed = imputer.fit_transform(X)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_imputed, y_binary, test_size=0.2, random_state=42) #random_state=42 is the

# Create and fit the Random Forest model
model_rf = RandomForestClassifier()
model_rf.fit(X_train, y_train)

# Print the accuracy level
accuracy = model_rf.score(X_test, y_test)
print("Model Accuracy:", str((accuracy*100).round(2)) + "%")

# Get the feature importances
importances = model_rf.feature_importances_
feature_names = X.columns.tolist()

# Create a DataFrame to store the results
```

```
importance_df = pd.DataFrame({'Feature': feature_names, 'Importance': importances})

# Sort the DataFrame by importance values
importance_df = importance_df.sort_values(by='Importance', ascending=False).reset_index(drop=True)

# Plot the feature importances
plt.figure(figsize=(15, 10))
plt.bar(importance_df['Feature'], importance_df['Importance'])
plt.xlabel('Features')
plt.ylabel('Importance')
plt.title('Feature Importances (Random Forest)')
plt.xticks(rotation=-45)
# plt.tight_layout()
plt.show()

# faeture importance do not have directions!
# To interpret the feature importance values in a Random Forest, you can focus on the relative rankings of the features. Feature
```

```
Model Accuracy: 86.84%
```


Feature Importances (Random Forest)
Importance
0.00
0.02
0.04
0.06
0.08
0.10
0.12
pct no internet
pct in poverty
pct no high school
pct no vehicle
pct non-hispanic black
pct non-hispanic asian
pct hispanic
pct non-hispanic white
pct unemployment
pct renter
pct pre-1980 housing
pct non-hispanic two
pct disability
pct ag workers
pct of limited English
pct non-hispanic nhpi
pct mobile homes
pct non-hispanic other
pct non-hispanic aian
Features

In [77]:
```
# print(len(feature_names))
```

In [29]:
```
# Using simplified data from PCA - accuracy dropped a bit
# MODEL 2: Random Forest Model
X = df_sel_dac.iloc[:,:-2]
y = df_sel_dac.iloc[:,-1]
y_binary = np.where(y == 'Y', 1, 0) #assign DAC as 1, otherwise 0
y_binary
# np.where(y_binary == 1)

#Impute missing values with the median value
imputer = SimpleImputer(strategy='median')
X_imputed = imputer.fit_transform(X)

# Perform PCA for dimensionality reduction
# pca = PCA(n_components=4) #83.08%
# pca = PCA(n_components=6) #84.34%
pca = PCA(n_components=8) #84.23%
# pca = PCA(n_components=0.95)  # Specify the desired % of variance preservation - this is 85.12%
X_pca = pca.fit_transform(X_imputed)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_pca, y_binary, test_size=0.2, random_state=42) #random_state=42 is the see

# Create and fit the Random Forest model
model_rf = RandomForestClassifier()
model_rf.fit(X_train, y_train)

# Print the accuracy level
accuracy = model_rf.score(X_test, y_test)
print("Model Accuracy:", str((accuracy*100).round(2)) + "%")

# Get the feature importances
importances = model_rf.feature_importances_ #this only includes the number of PCs, instead of the full list of original variable
feature_names = X.columns.tolist()

# check array lengths
print(len(importances))
print(len(feature_names))

# Create a DataFrame to store the results
# importance_df = pd.DataFrame({'Feature': feature_names, 'Importance': importances})

# # Sort the DataFrame by importance values
# importance_df = importance_df.sort_values(by='Importance', ascending=False).reset_index(drop=True)

# # Plot the feature importances
# plt.figure(figsize=(15, 10))
```

```
# plt.bar(importance_df['Feature'], importance_df['Importance'])
# plt.xlabel('Features')
# plt.ylabel('Importance')
# plt.title('Feature Importances (Random Forest)')
# plt.xticks(rotation=-45)
# # plt.tight_layout()
# plt.show()
```

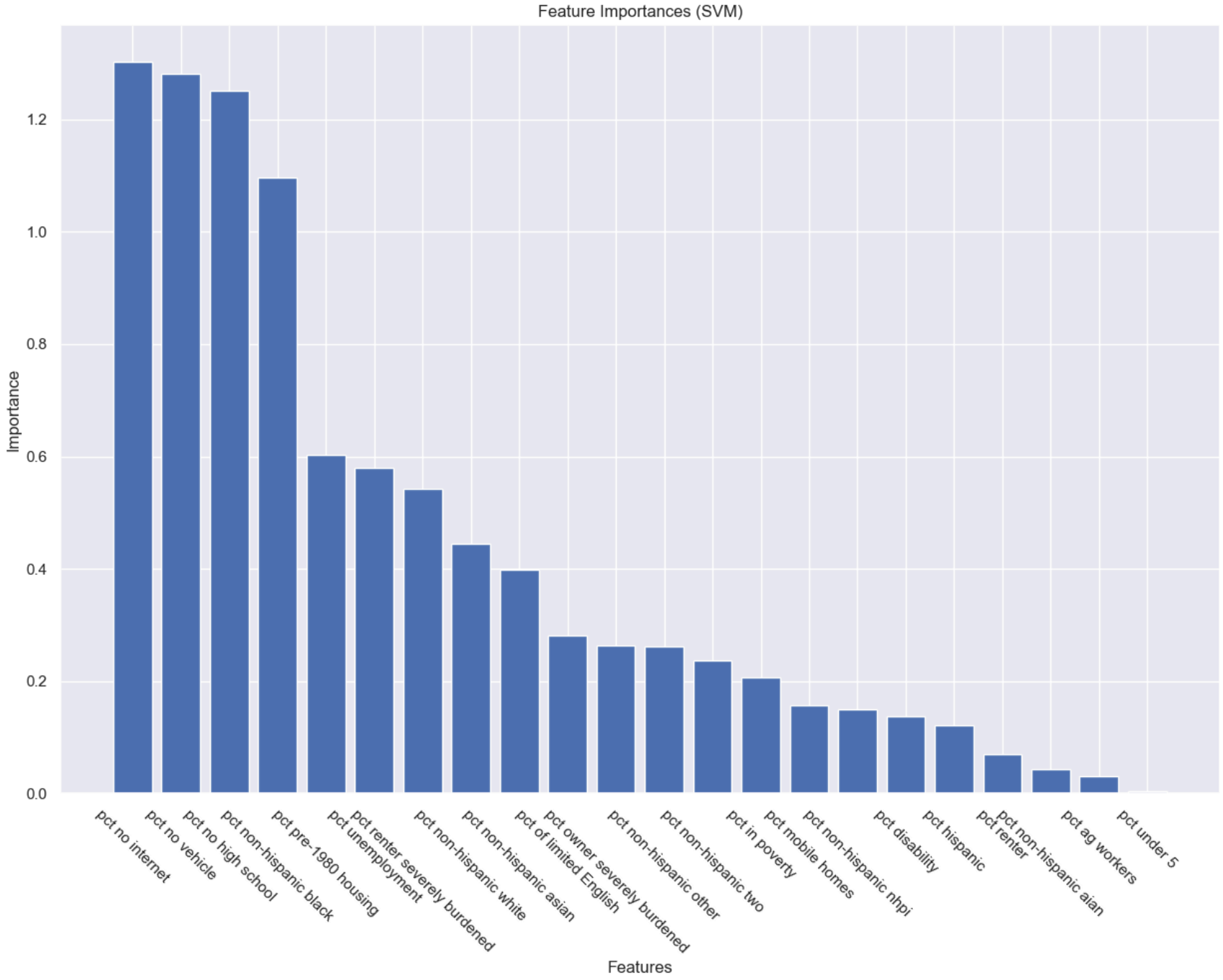
```
Model Accuracy: 86.84%
8
22
```

In [30]:

```
### MACHINE LEARNING (Classification) Models ###

# MODEL 3: SVM Model
X = df_sel_dac.iloc[:,:-2]
y = df_sel_dac.iloc[:,-1]
y_binary = np.where(y == 'Y', 1, 0) #assign DAC as 1, otherwise 0
y_binary
# np.where(y_binary == 1)

#Impute missing values with the median value
imputer = SimpleImputer(strategy='median')
X_imputed = imputer.fit_transform(X)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_imputed, y_binary, test_size=0.2, random_state=42) #random_state=42 is the

# Create and fit the SVM model
model = SVC(kernel='linear') #specify the linear kernel
model.fit(X_train, y_train)

# Print the accuracy level
accuracy = model.score(X_test, y_test)
print("Model Accuracy:", str((accuracy*100).round(2)) + "%")

# Get the feature importances (using coef_ for linear SVM)
importances = np.abs(model.coef_)[0]
feature_names = X.columns.tolist()

# # for non-linear kernels, Perform permutation importance
# perm_importance = permutation_importance(model, X_test, y_test, n_repeats=10, random_state=42)
# # Get feature importances from permutation importance results
# importances = perm_importance.importances_mean
# feature_names = X.columns.tolist()

# Create a DataFrame to store the results
importance_df = pd.DataFrame({'Feature': feature_names, 'Importance': importances})

# Sort the DataFrame by importance values
```

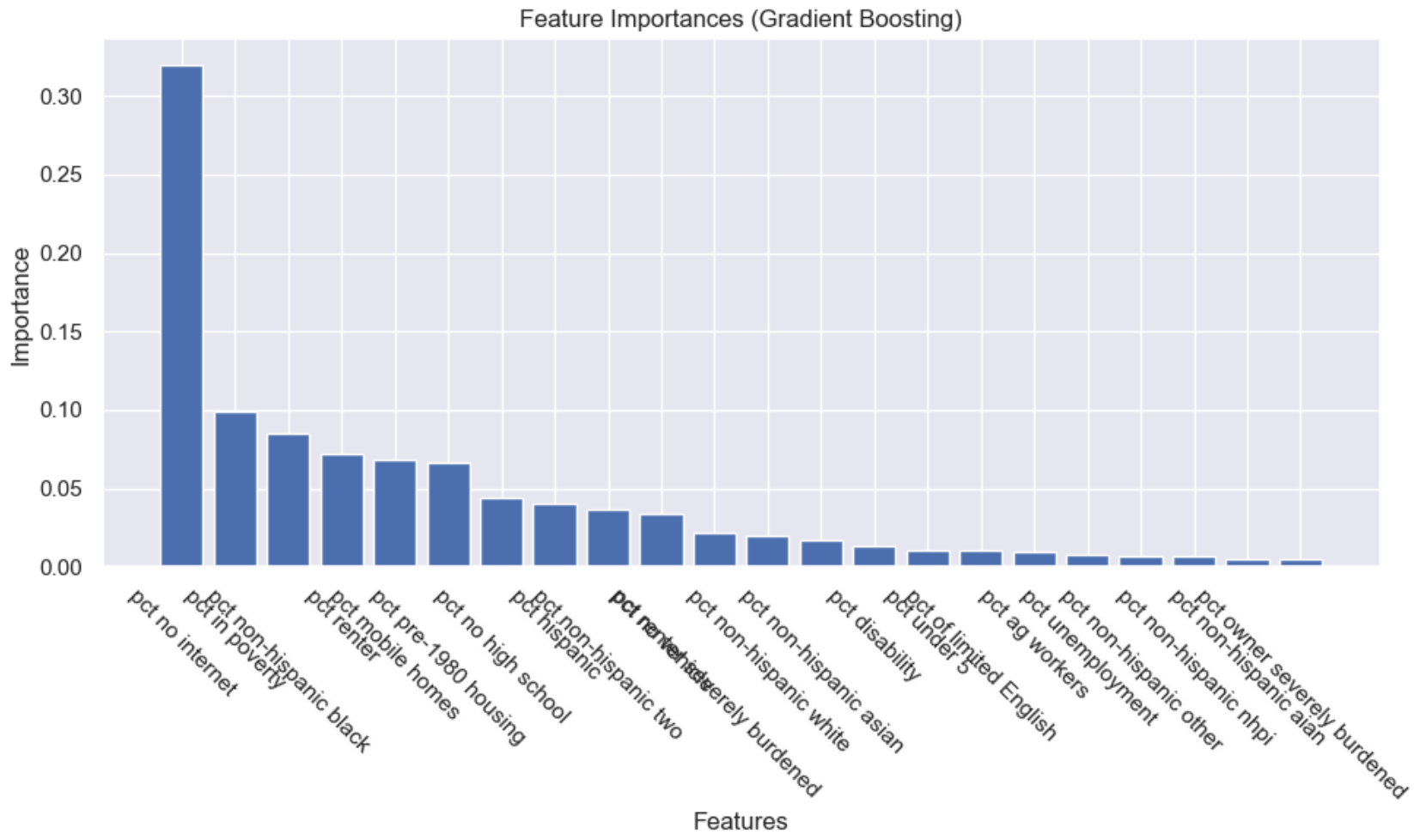
```
importance_df = importance_df.sort_values(by='Importance', ascending=False).reset_index(drop=True)

# Plot the feature importances
plt.figure(figsize=(15, 10))
plt.bar(importance_df['Feature'], importance_df['Importance'])
plt.xlabel('Features')
plt.ylabel('Importance')
plt.title('Feature Importances (SVM)')
plt.xticks(rotation=-45)
# plt.tight_layout()
plt.show()
```

```
Model Accuracy: 89.47%
```


Feature Importances (SVM)
Importance
0.0
0.2
0.4
0.6
0.8
1.0
1.2
pct no internet
pct no vehicle
pct no high school
pct non-hispanic black
pct pre-1980 housing
pct unemployment
pct renter severely burdened
pct non-hispanic white
pct non-hispanic asian
pct of limited English
pct owner severely burdened
pct non-hispanic other
pct non-hispanic two
pct in poverty
pct mobile homes
pct non-hispanic nhpi
pct disability
pct hispanic
pct renter
pct non-hispanic aian
pct ag workers
pct under 5
Features

In [31]:
```
### MACHINE LEARNING (Classification) Models ###

# MODEL 4: Gradient Boosting
X = df_sel_dac.iloc[:,:-2]
y = df_sel_dac.iloc[:,-1]
y_binary = np.where(y == 'Y', 1, 0) #assign DAC as 1, otherwise 0
y_binary
# np.where(y_binary == 1)

#Impute missing values with the median value
imputer = SimpleImputer(strategy='median')
X_imputed = imputer.fit_transform(X)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_imputed, y_binary, test_size=0.2, random_state=42) #random_state=42 is the

# Create and fit the Gradient Boosting model
model = GradientBoostingClassifier()
model.fit(X_train, y_train)

# Print the accuracy level
accuracy = model.score(X_test, y_test)
print("Model Accuracy:", str((accuracy*100).round(2)) + "%")

# Get the feature importances
importances = model.feature_importances_
feature_names = X.columns.tolist()

# Create a DataFrame to store the results
importance_df = pd.DataFrame({'Feature': feature_names, 'Importance': importances})

# Sort the DataFrame by importance values
importance_df = importance_df.sort_values(by='Importance', ascending=False).reset_index(drop=True)

# Plot the feature importances
plt.figure(figsize=(10, 6))
plt.bar(importance_df['Feature'], importance_df['Importance'])
plt.xlabel('Features')
plt.ylabel('Importance')
plt.title('Feature Importances (Gradient Boosting)')
plt.xticks(rotation=-45)
plt.tight_layout()
plt.show()
```

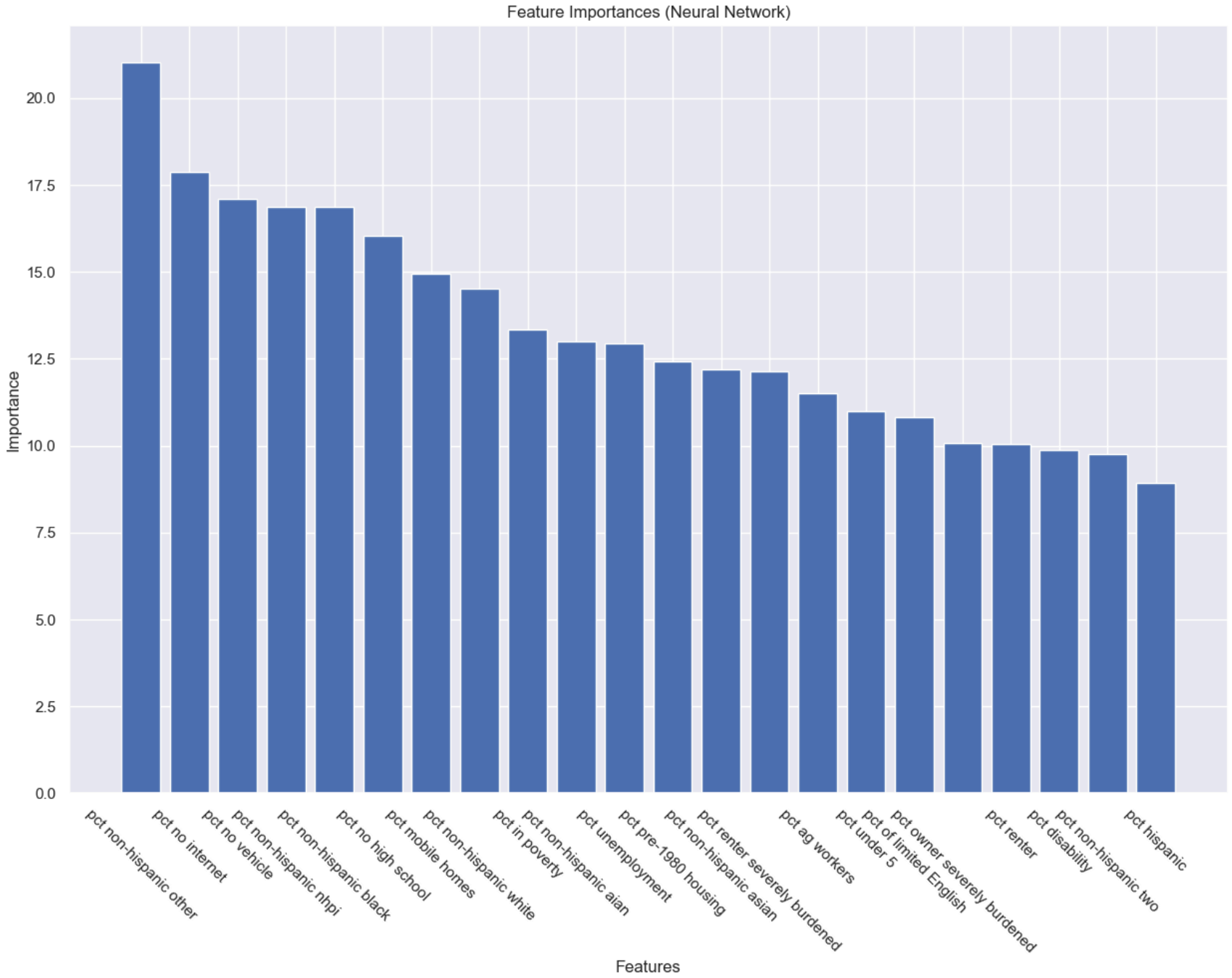
```
Model Accuracy: 86.84%
```



In [32]:
```
### MACHINE LEARNING (Classification) Models ###

# MODEL 5: Neural Networks
X = df_sel_dac.iloc[:,:-2]
y = df_sel_dac.iloc[:,-1]
y_binary = np.where(y == 'Y', 1, 0) #assign DAC as 1, otherwise 0
y_binary
# np.where(y_binary == 1)

#Impute missing values with the median value
imputer = SimpleImputer(strategy='median')
X_imputed = imputer.fit_transform(X)

# Split data into training and testing sets
```

```
X_train, X_test, y_train, y_test = train_test_split(X_imputed, y_binary, test_size=0.2, random_state=42) #random_state=42 is the

# Create and fit the Neural Networks model
model = MLPClassifier()
model.fit(X_train, y_train)

# Print the accuracy level
accuracy = model.score(X_test, y_test)
print("Model Accuracy:", str((accuracy*100).round(2)) + "%")

# Get the feature importances (using weights of the first layer)
importances = np.abs(model.coefs_[0]).sum(axis=1)
feature_names = X.columns.tolist()

# Create a DataFrame to store the results
importance_df = pd.DataFrame({'Feature': feature_names, 'Importance': importances})

# Sort the DataFrame by importance values
importance_df = importance_df.sort_values(by='Importance', ascending=False).reset_index(drop=True)

# Plot the feature importances
plt.figure(figsize=(15, 10))
plt.bar(importance_df['Feature'], importance_df['Importance'])
plt.xlabel('Features')
plt.ylabel('Importance')
plt.title('Feature Importances (Neural Network)')
plt.xticks(rotation=-45)
# plt.tight_layout()
plt.show()
```

```
/Users/wenhaowu/miniconda3/envs/gis/lib/python3.11/site-packages/sklearn/neural_network/_multilayer_perceptron.py:691: Convergen
ceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.
  warnings.warn(
Model Accuracy: 88.16%
```


Feature Importances (Neural Network)
Importance
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
pct non-hispanic other
pct no internet
pct no vehicle
pct non-hispanic nhpi
pct non-hispanic black
pct no high school
pct mobile homes
pct non-hispanic white
pct in poverty
pct non-hispanic aian
pct unemployment
pct pre-1980 housing
pct non-hispanic asian
pct renter severely burdened
pct ag workers
pct under 5
pct of limited English
pct owner severely burdened
pct renter
pct disability
pct non-hispanic two
pct hispanic
Features

In [33]:
```
### MACHINE LEARNING (Classification) Models ###

# MODEL 5 w/ PCA: Neural Networks
X = df_sel_dac.iloc[:,:-2]
y = df_sel_dac.iloc[:,-1]
y_binary = np.where(y == 'Y', 1, 0) #assign DAC as 1, otherwise 0
y_binary
# np.where(y_binary == 1)

#Impute missing values with the median value
imputer = SimpleImputer(strategy='median')
X_imputed = imputer.fit_transform(X)

pca = PCA(n_components=0.95)  # Specify the desired % of variance preservation - this is 85.12%
X_pca = pca.fit_transform(X_imputed)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_pca, y_binary, test_size=0.2, random_state=42) #random_state=42 is the see

# Create and fit the Neural Networks model
model = MLPClassifier()
model.fit(X_train, y_train)

# Print the accuracy level
accuracy = model.score(X_test, y_test)
print("Model Accuracy:", str((accuracy*100).round(2)) + "%")

# # Get the feature importances (using weights of the first layer)
# importances = np.abs(model.coefs_[0]).sum(axis=1)
# feature_names = X.columns.tolist()

# # Create a DataFrame to store the results
# importance_df = pd.DataFrame({'Feature': feature_names, 'Importance': importances})

# # Sort the DataFrame by importance values
# importance_df = importance_df.sort_values(by='Importance', ascending=False).reset_index(drop=True)

# # Plot the feature importances
# plt.figure(figsize=(15, 10))
# plt.bar(importance_df['Feature'], importance_df['Importance'])
# plt.xlabel('Features')
# plt.ylabel('Importance')
# plt.title('Feature Importances (Neural Network)')
# plt.xticks(rotation=-45)
# # plt.tight_layout()
# plt.show()
```

```
Model Accuracy: 88.16%
```

```
/Users/wenhaowu/miniconda3/envs/gis/lib/python3.11/site-packages/sklearn/neural_network/_multilayer_perceptron.py:691: Convergen
ceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.
  warnings.warn(
```

In [83]:
```
# The Multi-Layer Perceptron (MLP) model is a type of neural network that is designed to learn patterns and make predictions bas

# The MLP model works in a sequential manner, passing the input data through each layer, where each layer performs certain compu

# During the training phase, the MLP model adjusts the weights of the connections between neurons to minimize the difference bet

# Once the MLP model is trained, it can be used to make predictions on new, unseen data. It takes the input data, passes it thro

# In summary, the MLP model performs computations on input data through interconnected layers of artificial neurons. By adjustin
```

In [35]:
```
### Predict the entire state's DACs using the selected model
# MODEL selected: Random Forest Model
X = df_sel_dac.iloc[:,:-2]
y = df_sel_dac.iloc[:,-1]
y_binary = np.where(y == 'Y', 1, 0) #assign DAC as 1, otherwise 0
y_binary

#Impute missing values with the median value
imputer = SimpleImputer(strategy='median')
X_imputed = imputer.fit_transform(X)

# Create and fit the Random Forest model
model_rf = RandomForestClassifier()
model_rf.fit(X_imputed, y_binary)

#predict
predictions = model_rf.predict(X_imputed)
print("predictions: ", predictions)
print("predictions w/ DAC: ", np.count_nonzero(predictions==1))

#convert pred to df
pred_df = pd.DataFrame({'pred_DAC': predictions})
pred_df.head(10)
```

```
predictions:  [0 0 0 0 0 0 0 0 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0
 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0]
predictions w/ DAC:  47
```

Out[35]:

	pred_DAC
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	1
9	1

In [ ]:

Mapping the Prediction Results

In [36]:
```
# Join to GIS shapefile for mapping
df_sel_dac_pred = pd.concat([df_sel_dac,pred_df],axis=1)
df_sel_dac_pred.head(5)
# df_sel_dac.merge(predictions,how='inner',left_on='GEOID',right_on='GEOID_2')
```

Out[36]:

	pct under 5	pct disability	pct ag workers	pct no high school	pct of limited English	pct in poverty	pct unemployment	pct renter severely burdened	pct owner severely burdened	pct non-hispanic white	pct non-hispanic black	pct non-hispanic aian	pct non-hispanic asian	pct non-hispanic nhpi
0	0.040685	0.056845	0.005484	0.021808	0.015944	0.043719	0.053584	0.000000	0.203200	0.689813	0.052310	0.000000	0.145916	0.000000
1	0.078714	0.036977	0.000000	0.028736	0.019717	0.074988	0.067186	0.072539	0.102041	0.698649	0.017699	0.004192	0.116907	0.000000
2	0.022602	0.048657	0.002985	0.040097	0.022396	0.070815	0.077107	0.111768	0.055966	0.585513	0.094145	0.004627	0.113009	0.000000
3	0.075269	0.096821	0.004334	0.023963	0.014914	0.099148	0.000000	0.192785	0.100802	0.640486	0.116643	0.005376	0.087190	0.000000
4	0.040010	0.069544	0.000000	0.021573	0.027961	0.103781	0.081315	0.256264	0.300874	0.414789	0.251456	0.001013	0.093695	0.001519

In [37]:
```
n = len(df_sel_dac)
print("total records:",n) #total records

#how many actual DACs are predicted not to be non-DAC?
FN = np.count_nonzero((df_sel_dac_pred['SB535_DAC']=='Y') & (df_sel_dac_pred['pred_DAC']==0))
print("False Negatives:",FN)

#how many actual non-DACs are predicted to be DAC?
FP = np.count_nonzero((df_sel_dac_pred['SB535_DAC']=='N') & (df_sel_dac_pred['pred_DAC']==1))
print("False Positives:",FP)

#prediction accuracy
pred_accuracy = 1-(FN+FP)/n
print("Prediction Accuracy:", str(round(pred_accuracy*100,2)) + "%")
```

```
total records: 379
False Negatives: 0
False Positives: 0
Prediction Accuracy: 100.0%
```

In [38]:
```
# ### Read GIS data ###
ca_ct['GEOID_str']=ca_ct['GEOID'].astype(float).astype(str).str[:-2]
ca_ct.head(2)
```

Out[38]:

	STATEFP	COUNTYFP	TRACTCE	GEOID	NAME	NAMELSAD	MTFCC	FUNCSTAT	ALAND	AWATER	INTPTLAT	INTPTLON	geometry
0	06	037	137504	06037137504	1375.04	Census Tract 1375.04	G5020	S	3837562	0	+34.1480383	-118.5720594	POLYGON ((-118.58119 34.14318, -118.58099 34.1...
1	06	037	138000	06037138000	1380	Census Tract 1380	G5020	S	4472196	0	+34.1488008	-118.5910495	POLYGON ((-118.60573 34.14585, -118.60561 34.1...

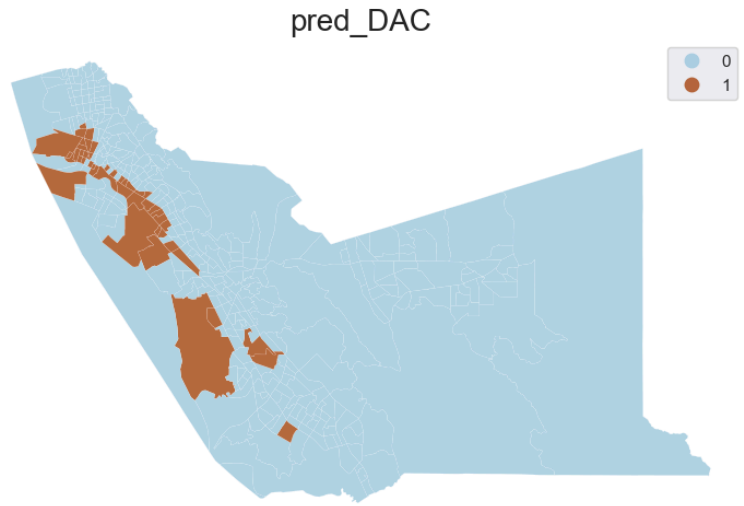
In [39]:
```
#join prediction to GIS layer
ca_ct_dac_pred = ca_ct.merge(df_sel_dac_pred,how='inner',left_on='GEOID_str',right_on='GEOID_Copy')
ca_ct_dac_pred.head(3)
```

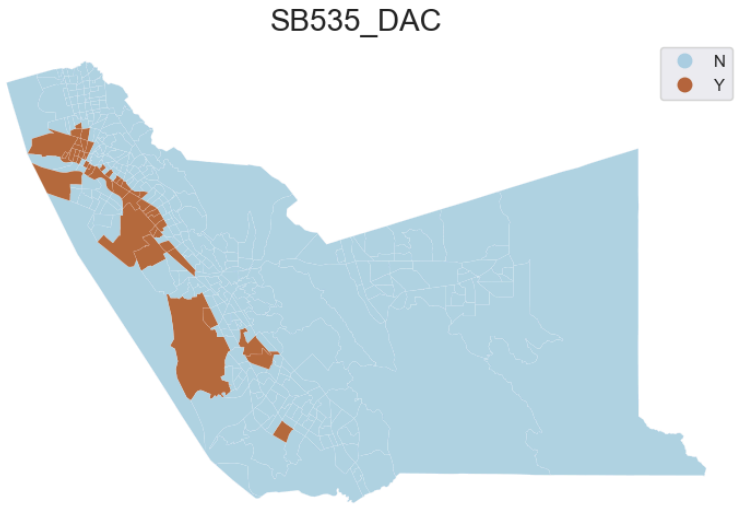
Out[39]:

	STATEFP	COUNTYFP	TRACTCE	GEOID	NAME	NAMELSAD	MTFCC	FUNCSTAT	ALAND	AWATER	INTPTLAT	INTPTLON	geometry
0	06	001	441402	06001441402	4414.02	Census Tract 4414.02	G5020	S	1662787	0	+37.5823293	-122.0400088	POLYGON ((-122.05035 37.58349, -122.05019 37.5...
1	06	001	441501	06001441501	4415.01	Census Tract 4415.01	G5020	S	2858378	0	+37.5695603	-122.0740416	POLYGON ((-122.09555 37.56607, -122.09516 37.5...
2	06	001	441521	06001441521	4415.21	Census Tract 4415.21	G5020	S	1623727	0	+37.5730234	-122.0528477	POLYGON ((-122.06043 37.57489, -122.06034 37.5...

In [40]:
```
#mapping prediction result
fig,axes = plt.subplots(ncols=2,figsize=(20,15))

mapname1 = 'pred_DAC'

ca_ct_dac_pred.plot(column=mapname1,categorical=True,cmap='Paired',
                    figsize=(10,15),alpha=0.9,linewidth=.1,
                    ax=axes[0],
                    edgecolor=None,legend=True)
#colormaps: https://matplotlib.org/stable/tutorials/colors/colormaps.html
```

```
axes[0].set_axis_off();
axes[0].set_title(mapname1,fontdict={
    'fontsize':20,
    'fontweight':3
})

# plt.savefig(f"{mapname1}.png",dpi=200)

mapname2 = 'SB535_DAC'
ca_ct_dac_pred.plot(column=mapname2,categorical=True,cmap='Paired',
                            figsize=(10,15),alpha=0.9,linewidth=.1,
                            ax=axes[1],
                            edgecolor=None,legend=True)
#colormaps: https://matplotlib.org/stable/tutorials/colors/colormaps.html

axes[1].set_axis_off();
axes[1].set_title(mapname2,fontdict={
    'fontsize':20,
    'fontweight':3
})

plt.savefig(f"{mapname1}+'-'+{mapname2}.png",dpi=200)
```





Here in the following cell, go to the "Cell" button, Click the "Run All Above" to run the entire script, and look for the output CSVs and shapefiles in the script folder:

W:\Templates + Work Samples\Data Analytics\RA_DataAnalytics\SVI_Script

In [ ]:

In [175…
```
#DRAFT
# histograms
# sb.set(style='darkgrid')
# fig,ax = plt.subplots(ncols=3,nrows=7,figsize=(15,35))

# sb.histplot(data=df_sel,x='pct under 5',kde=True,ax=ax[0,0])
# sb.histplot(data=df_sel,x='pct disability',kde=True,ax=ax[0,1])
# sb.histplot(data=df_sel,x='pct ag workers',kde=True,ax=ax[0,2])

# sb.histplot(data=df_sel,x='pct no high school',kde=True,ax=ax[1,0])
# sb.histplot(data=df_sel,x='pct in poverty',kde=True,ax=ax[1,1])
# sb.histplot(data=df_sel,x='pct unemployment',kde=True,ax=ax[1,2])

# sb.histplot(data=df_sel,x='pct renter severely burdened',kde=True,ax=ax[2,0])
# sb.histplot(data=df_sel,x='pct owner severely burdened',kde=True,ax=ax[2,1])
# sb.histplot(data=df_sel,x='pct renter',kde=True,ax=ax[2,2])

# sb.histplot(data=df_sel,x='pct pre-1980 housing',kde=True,ax=ax[3,0])
# sb.histplot(data=df_sel,x='pct mobile homes',kde=True,ax=ax[3,1])
# sb.histplot(data=df_sel,x='pct no vehicle',kde=True,ax=ax[3,2])

# sb.histplot(data=df_sel,x='pct no internet',kde=True,ax=ax[4,0])
# sb.histplot(data=df_sel,x='pct hispanic',kde=True,ax=ax[4,1])
# sb.histplot(data=df_sel,x='pct non-hispanic white',kde=True,ax=ax[4,2])

# sb.histplot(data=df_sel,x='pct non-hispanic black',kde=True,ax=ax[5,0])
# sb.histplot(data=df_sel,x='pct non-hispanic aian',kde=True,ax=ax[5,1])
# sb.histplot(data=df_sel,x='pct non-hispanic asian',kde=True,ax=ax[5,2])

# sb.histplot(data=df_sel,x='pct non-hispanic nhpi',kde=True,ax=ax[6,0])
# sb.histplot(data=df_sel,x='pct non-hispanic other',kde=True,ax=ax[6,1])
# sb.histplot(data=df_sel,x='pct non-hispanic two',kde=True,ax=ax[6,2])



# Calculate Pearson correlation coefficient
# pearson_corr = pairwise.pairwise_distances(X, metric="correlation")

# Print the correlation matrix
# print("Pearson Correlation Matrix:")
# print(pearson_corr)
```