Data Bootcamp Project

Analysis of Housing and Income Census Data for New York City

Riccardo Maltese (rm5124) **Ludovica Diroma (ld2240)**

Introduction

This project has the objective of analyzing income and housing data provided by the United States Census for the District of New York City in order to see the presence of any patter on the territory of selected census variables, such as Median Household Income, Median Contract Rent, Median House Value and Tenure.

The data will be retrieve thorugh the Census API as well as a Shapefile of New York City found on the NYC Planning website. Due to the timeline of census data publication, this project will focus on the data for the year 2016.

The project has the second purpose of enabling any user to easily adoperate the code provided to analyze any kind of data from the census for the District of New York. For this reason, we invite anyone to freely change the variables analyzed by us with any other which suits their interests.

As a matter of usability, we also suggest to change the datapath used to read the Shapefile present in the project to the chosen location, as well as obtain a personal Census API key to be used for the retrival of the data from their database.

As a last disclaimer, due to the presence of non standard libraries, we would suggest to download and install the following packages before running the notebook: seaborn, census, us, folium geopandas and branca.

Notebook Setup

The following code is meant to import the necessary libraries used in this project.

pandas will be used to create and work on a series of Dataframes containing the Census data and the Shapefiles used to plot the data obtained.

seaborn is a library allowing the user to create more pleasing grapichal plots than the standard ones provided by matplotlib, hence it will be used just for a design purposes.

census is the python library provided by the US Census to access their API and retrieve historical public census data. This package will be used to obtain the data on the New York City Boroughs we will analyze.

us is a handy library which enable to easily retrieve us states information. We will use it to speed up the retrival process of the census data.

folium is a library that can be used to create beautiful customizable interactive maps. We Will use it to plot our data through an interactive choropleth map.

geopandas eases the use of geospatial data with python. We will implement it to treat our Shapefiles in order to convert them to GeoDataframe format and use them to create the interactive map.

branca is a library born from a spinoff of folium in order to better manage all the styling feature use in the latter package. We will import colormap in order to create automated colormaps which will fit our datasets.

matplotlib will be used to plot our data and visually analyze any potential patter in the dataset which may help us study the relation with the different census variables.

statsmodel will be used primarily to study the correlation between the various census data variables obtained from the US Census API.

In [1]:
```
import pandas as pd
import seaborn as sns
from census import Census
from us import states
import folium
from folium import plugins
import geopandas
import branca.colormap as cml
import matplotlib.pyplot as plt
import statsmodels.api as sm

sns.set(color_codes=True)
%matplotlib notebook
```

Variables Declaration

Since we will need to work on data retrieved through the Census API, we first need to setup the access key which will authorize the connection to the Census database.

In order to do this, we need to store our API key in a string type variable , as well as declare another variable containing the census library function to authorize our access.

In [2]:
```
# CENSUS API KEY SETUP
CENSUS_API_KEY = '6bf43dc37b1f5c7600a05bab878386d833ac9e95'
c = Census(CENSUS_API_KEY)
```

Before starting to work on the data, it is necessary to also set up the infomation we will use to retrieve the dataset we need from the census database.

In order to do so, we decided to declare two different dictionaries containing such items.

From what you can se in the following code, the first dictionary, called **NYC_BOROUGHS**, has the names of the 5 boroughs of New York as keys, while the corresponding values are lists containing each the associated FIPS code and the identificative number used in the Shapefiles we will work with. The second dictionary declared, instead called **census_data**, has the task of containing the census variables we wish to analyze. For such reason, all the keys correspond to the variables' codes used by the Census, while the values correspond to the human readable names.

In [3]:
```
# Dictionary containing New York boroughs as keys and correspoding list with FI
PS code and Shapefile
# identification number as values
```

```
NYC_BOROUGHS = {'manhattan': ['061', '1'],
                'bronx': ['005', '2'],
                'brooklyn': ['047', '3'],
                'queens': ['081', '4'],
                'statenisland': ['085', '5']}

# Dictionary containing the census variables selected with the variables codes
 as keys and the corresponding
# names as values
census_data = {'B19013_001E': 'Median Household Income',
               'B25058_001E': 'Median Contract Rent',
               'B25077_001E': 'Median House Value',
               'B25003_001E': 'Tenure: Total',
               'B25003_002E': 'Tenure: Owned',
               'B25003_003E': 'Tenure: Rented'}
```

For sake of code organization and division, all the necessary lists and dictionaries used in the project have been defined before starting to work with the data.

This includes the list declared below, which contains all the possible colorstyles applicable to the choropleth map we will create later.

In [4]:
```
# List containing the codes for the colorstyles used in the choropleth map
colorstyles = ['Blues_09', 'BuGn_09', 'BuPu_09', 'GnBu_09', 'Greens_09', 'Greys
_09', 'Oranges_09', 'OrRd_09',
               'PuBu_09', 'PuBuGn_09', 'PuRd_09', 'Purples_09', 'RdPu_09', 'Red
s_09', 'YlGn_09', 'YlGnBu_09',
               'YlOrBr_09','YlOrRd_09']
```

Functions Declaration

After realizing that working with the number of variables provided by the Census can create an enormous amount of redundant and repetitive code, it has been decided to optimize the overall process of data formatting and analysis we will implement by declaring all the necessary functions substituting hundreds of unnecessary lines of code.

The following functions are hence representing the entire data retriaval, formatting and analysis we needed for this project.

CENSUS_DATA:

This function has the task to return the data from the US Census database for the variable the user chooses, following the FIPSCODE values contained in the *NYC_BOROUGHS* dictionary.

The function ***c.acs5.state_county_tract*** is provided by the *census* library.

In [5]:
```
# returns the data from the US Census database for the variable chosen followin
g the FIPSCODE values
# given.
def CENSUS_DATA(variable, FIPSCODE):
    return c.acs5.state_county_tract(variable,  states.NY.fips, FIPSCODE, Censu
s.ALL)
```

CENSUS_DF_FM:

This function has the task to apply all the necessary formatting to the dataframe df passed and return the formatted dataframe for the variable var chosen.

We start by creating an list called index containing the values belonging to the *census_data* which will be used to rename the columns in the dataframe to a readable name, since the census variables in the dataframe retrieved from the Census API are identified by code. The other columns we rename are the ones we will later need for the analysis and the plotting of the data.

For comprehension purposes, we then sort the dataframe by census tract, and then we proceed at droopping the 'State' column, since we downloaded data at a tract level and state identification is just redundant.

To finish, we reset the index and drop the redundant one for data cleanliness, in order to have a tidy and readable dataframe readily available to the user.

In [6]:
```
# returns the formatted dataframe df chosen for the variable var chosen.
def CENSUS_DF_FM(df, var):
    index = list(census_data.values()).index(var)
    for x in df:
        df[x] = df[x].rename(index=str, columns={list(census_data.keys())[index
]: var,
                                                 'NAME': 'Name',
                                                 'county': 'BoroName',
                                                 'state': 'State',
                                                 'tract': 'CT2010'
                                                })
        df[x] = df[x].sort_values('CT2010')
        df[x] = df[x].drop(labels=['State'], axis=1)
        df[x] = df[x].reset_index()
        df[x] = df[x].drop(labels='index', axis=1)
    return df
```

SHP_FM:

This function returns the formatted geo dataframe df containing the shapefiles we will use to plot the choropleth map.

In order to be able to plot the data we will retrieve, it is necessary to format the dataframe containing the Shapefiles in a way that may let us work seemlesly between this dataframe and the one containing the Census data. For this reason, we eliminated all the redundant data which may block us from make the tue dataframes compatible: this means slicing the Shapefiles' dataframe so as to drop all the unnecessary data related to census tracts we will not use, like parks, cemetery, or any other area which may have a zero value variable, like population or household income. This process is done by inserting the desired column variable from the Census dataframe we will create into the corresponding Shapefile dataframe.

In [7]:
```
# returns the formatted geo dataframe df containing the shapefiles used to plot
 the choropleth map
def SHP_FM(df):
    for x in df:
        for y in df[x]:
            df[x][y] = df[x][y].reset_index()
            df[x][y] = df[x][y].drop(labels='index', axis=1)

            df_nyc = list(set(df[x][y]['CT2010'].unique()).difference(set(NYC_D
ATA[x][y]['CT2010'].unique())))
            nyc_df = list(set(NYC_DATA[x][y]['CT2010'].unique()).difference(set
(df[x][y]['CT2010'].unique())))
            if len(df_nyc)==0:
                NYC_DATA[x][y] = NYC_DATA[x][y].loc[~NYC_DATA[x][y]['CT2010'].i
sin(nyc_df)]
            else:
                df[x][y] = df[x][y].loc[~df[x][y]['CT2010'].isin(df_nyc)]
```

```
            df[x][y] = df[x][y].set_index(NYC_DATA[x][y].index)
            df[x][y].insert(3, x, NYC_DATA[x][y][x])
            df[x][y] = df[x][y].loc[~df[x][y]['NTAName'].str.lower().str.contai
ns('park-cemetery-etc-'),:]
            df[x][y] = df[x][y].loc[(df[x][y][x]>0)]
    return df
```

style_function:

This function returns a customizable conditional style function depending on the values of the data plotted.

In order to properly plot the choropleth maps of the data we are going to analyze, it is necessary to fit the color fills following a conditional function depending from the disrtribution of the data analyzed. Since manually estimating the distribution and quartiles of our data would be extremely time consuming, we opted to create this function which is able to automatically distribute the colorscale following the distribution of our data.

In [8]:
```
# returns a customizable conditional style function depending on the values of
 the data plotted
def style_function(df, var, color):
    colorscale = eval('cml.linear.{}.scale(df[var].describe()[3], df[var].descr
ibe()[7])'.format(color))

    color_function = lambda x: {'fillOpacity': 0.8,
                                'weight': 1,
                                'color' : 'black',
                                'fillColor': '#black' if x['propertie
s'][var] is None else
                                colorscale(x['properties'][var])
                               }
    return color_function
```

colorscale:

This function returns a colorscale legend to be added to the map depending on the data plotted.

Since it would be difficult to read the choropleth map without a legend with the colorscale created, we added this function which create the mentioned colorscale legend for the user.

In [9]:
```
# returns a colorscale legend to be added to the map depending on the data plot
ted.
def colorscale(set_caption, df, var, color):
    colorscale = eval('cml.linear.{}.scale(df[var].describe()[3], df[var].descr
ibe()[7])'.format(color))
    colorscale.caption = set_caption
    return colorscale
```

MAP:

This function return a folium interactive map adapted to the data chosen following a colorstyle picked from the colorstyles list.

This is the main function for the creation of the choroplet maps we will need to plot the data. The function has been created with the possibility of plotting two different datasets, in order to be able to compare them visually. However, given the processing memory needed to run these maps and also the fact that it would be preferrable to plot all the dataset alayzed, it is suitable to avoid plotting multiple census variables in the same map. Hence, three additional optional variables have been created, two equal to the main ones used in this function, and the third used to activate the double choropleth map, for the user to be able to freely try this function.

The function mainly relies on items belonging to the *folium* library, which presents analogies with the matplotlib library in term of processes to plot visual elements. We start by declaring a map item with coordinates corresponding to New York City, giving also a background map style. We then proceed to the GeoJson we will create containing the data necessary to plot the polygons' lines composing the census tracts. Then, for readability reasons, we implemented a interactive tooltip which displays the block name and corresponding variable value when hovered over with the cursor, as well as the colormap legend created before with the *colormap* function.

For visualization optimizaiton, it has also been added the automatic detection of the retina display, as well as the full screen toogle function, which may increase the resolution of the map and give the possibility to look with a higher precision the choropleth map.

In [10]:
```
# return a folium interactive map adapted to the data chosen following a colors
tyle chosen from the colorstyles
# list
def MAP(var1, colorstyle1, var2=0, colorstyle2=0, secondarymap=False):
    NYC_map = folium.Map(location=[40.738, -73.98],
                         zoom_start=10.5,
                         tiles='CartoDBPositron',
                          control_scale=True)

    map1 = folium.GeoJson(nyc_shp[var1],
                          name='NYC {}'.format(var1),
                          style_function = style_function(nyc_shp[var1], var1,
colorstyle1),
                          tooltip=folium.features.GeoJsonTooltip(fields=['BoroN
ame', 'NTAName', var1],
                                                                 aliases=['Boro
ugh', 'Block', '{}'.format(var1)]))
    map1.add_to(NYC_map)
    colormap1 = colorscale('{}'.format(var1),nyc_shp[var1], var1, colorstyle1)
    colormap1.add_to(NYC_map)
    if secondarymap==True:
        map2 = folium.GeoJson(nyc_shp[var2],
                              name='NYC {}'.format(var2),
                              style_function = style_function(nyc_shp[var2], va
r2, colorstyle2),
                              tooltip=folium.features.GeoJsonTooltip(fields=['N
TAName', var2],
                                                                     aliases=[
'Block', '{} ($)'.format(var2)]))
        map2.add_to(NYC_map)

    folium.LayerControl().add_to(NYC_map)
    plugins.Fullscreen(position='topright', force_separate_button=True).add_to(
NYC_map)

    NYC_map

    return NYC_map
```

Data Retrieval & Manupulation

This is the starting point of our analysis. In this section we will retrieve the necessary data from the United State Census and load the Shapefiles containing the data to plot the census tract map of New York City.

We start by creating through the *geopandas* library a GeoDataframe from the Shapefile 'nyct2010.shp' which contains all the infomation about the geometries of the census blocks of New York. Since the DataFrame created contains information which we do not care about for our goal, we

slice it and keep only the columns containing the necessary variables, and for readability purposes we sort it by the census block code, as we did before in the function to format the datafram we will create with the census data.

In [11]:
```
# New York City Boroughs Map Shapefiles Dataframe
NY_shp = geopandas.GeoDataFrame.from_file('nyct2010_18c/nyct2010.shp')
NY_shp = NY_shp[['BoroCode', 'BoroName', 'NTAName', 'CT2010', 'Shape_Leng', 'Shape_Area', 'geometry']]
NY_shp = NY_shp.sort_values('CT2010')
```

As we can see, the dataframe containing the Shapefiles contains different variables:

- BoroCode contains the identification code for the New York City borough, ranging from 1 to 5, as we stated in the dictionary *NYC_BOROUGHS* at the beginning. This will help us distinguish the boroughs.
- BoroName contains the exact names of the five Boroughs.
- NTAName represents the block name, which usually corresponds to the neighbourhood (or neighbourhoods) to which the variable belongs to.
- CT2010 indicates the census tract code.
- Shape_Leng, Shape_area and geometry are the three variables necessary to plot the plygons on our map.

In [12]:
```
NY_shp
```

Out[12]:

	BoroCode	BoroName	NTAName	CT2010	Shape_Leng	Shape_Area	geome
1767	2	Bronx	Rikers Island	000100	18898.116621	1.816383e+07	POLYG ((1019454.697021 225654.328796386
1956	4	Queens	Hunters Point-Sunnyside-West Maspeth	000100	39868.188978	1.227737e+07	POLYG ((998348.4624023 214174.082641601
814	1	Manhattan	park-cemetery-etc-Manhattan	000100	11023.047911	1.844421e+06	(POLYG ((972081.7880249 190733.467041015
1283	3	Brooklyn	Brooklyn Heights-Cobble Hill	000100	6527.277710	2.237278e+06	POLYG ((986764.348815 194584.62438964
1958	2	Bronx	Soundview-Castle Hill-Clason Point-Harding Park	000200	15610.702268	5.006558e+06	POLYG ((1023972.527832 232680.658386230
380	4	Queens	Woodhaven	000200	5114.375159	1.665497e+06	POLYG ((1022155.474182 191406.320800781
1233	3	Brooklyn	Sunset Park West	000200	9075.284871	2.952085e+06	POLYG ((982507.2609863 178067.092590332
1895	1	Manhattan	Lower East Side	000201	4748.325960	9.715994e+05	POLYG ((988548.2182006 197770.374816894
1858	1	Manhattan	Lower East	000202	8568.089235	3.315114e+06	(POLYG

			Side				((989137.1102294 196325.438781738
1614	5	Staten Island	West New Brighton-New Brighton-St. George	000300	16914.427461	4.119894e+06	POLYG ((964105.8546142 171704.149414062
2006	3	Brooklyn	Brooklyn Heights-Cobble Hill	000301	6466.975191	2.142136e+06	POLYG ((985832.7232055 193771.676635742
1959	2	Bronx	Soundview-Castle Hill-Clason Point-Harding Park	000400	24725.471690	8.561175e+06	POLYG ((1026849.274230 235548.773986816
1583	4	Queens	Woodhaven	000400	6508.267929	2.420866e+06	POLYG ((1022701.828186 190216.903991699
1768	1	Manhattan	park-cemetery-etc-Manhattan	000500	32407.898603	9.081808e+06	(POLYG ((981219.0557861 188655.315795898
19	3	Brooklyn	Brooklyn Heights-Cobble Hill	000501	4828.170042	9.015070e+05	POLYG ((986185.986999 193380.880187988
1653	3	Brooklyn	Brooklyn Heights-Cobble Hill	000502	4866.833820	1.038567e+06	POLYG ((986643.5615844 193217.478393554
1886	1	Manhattan	Lower East Side	000600	6970.634699	2.583418e+06	POLYG ((986961.185607 199553.643188476
2032	5	Staten Island	Stapleton-Rosebank	000600	22538.430260	7.195798e+06	POLYG ((968486.5178222 160755.694213867
1580	4	Queens	Woodhaven	000600	6437.134212	2.307341e+06	POLYG ((1022946.834228 188809.537841796
2164	5	Staten Island	West New Brighton-New Brighton-St. George	000700	11383.809763	5.500153e+06	POLYG ((963725.897399 174346.325195312
1957	4	Queens	Hunters Point-Sunnyside-West Maspeth	000700	10784.721024	5.867575e+06	POLYG ((998318.485412 209801.854003906
2007	3	Brooklyn	Brooklyn Heights-Cobble Hill	000700	6313.472356	1.927179e+06	POLYG ((985160.395385 191915.109008789
1685	1	Manhattan	Battery Park City-Lower Manhattan	000700	9802.917997	2.442012e+06	(POLYG ((984804.689392 193867.306396484
1847	1	Manhattan	Chinatown	000800	6362.785762	2.366871e+06	POLYG ((986494.144592 197775.830017089
1854	5	Staten Island	Stapleton-Rosebank	000800	14770.527758	8.555207e+06	POLYG ((967460.874023 160212.181396484
1581	4	Queens	Woodhaven	000800	5918.323526	2.043534e+06	POLYG

							((1024508.3074340 189400.563598632
1855	1	Manhattan	Battery Park City-Lower Manhattan	000900	16484.098901	3.122059e+06	(POLYG ((984032.8836059 192223.748413085
1160	3	Brooklyn	Brooklyn Heights-Cobble Hill	000900	5883.375467	1.737777e+06	POLYG ((986838.3345947 192326.738586425
0	5	Staten Island	West New Brighton-New Brighton-St. George	000900	7729.016794	2.497010e+06	POLYG ((962269.1260375 173705.500183105
381	4	Queens	Woodhaven	001000	5286.510472	1.743090e+06	POLYG ((1024131.752990 190606.958984375
...	...	...	...	...	...	...	
2023	4	Queens	Auburndale	141700	15257.113981	9.659754e+06	POLYG ((1043448.557617 211332.553222656
979	4	Queens	Auburndale	142900	13094.027239	5.720441e+06	POLYG ((1045058.744628 213811.563781738
602	4	Queens	Bayside-Bayside Hills	143500	8640.845905	3.803289e+06	POLYG ((1049481.478393 213544.225219726
908	4	Queens	Bayside-Bayside Hills	144100	8947.044079	3.627720e+06	POLYG ((1050900.208984 213488.715209960
744	4	Queens	Bayside-Bayside Hills	144700	8532.440279	4.401155e+06	POLYG ((1048446.359191 215616.461791992
980	4	Queens	Auburndale	145101	6870.722823	2.704350e+06	POLYG ((1044262.279785 215421.085021972
580	4	Queens	Bayside-Bayside Hills	145102	9156.789096	3.780662e+06	POLYG ((1046366.501220 213793.676635742
2021	4	Queens	Auburndale	145900	8558.382275	4.007243e+06	POLYG ((1042804.763000 213099.012634277
809	4	Queens	Bayside-Bayside Hills	146300	7523.875902	3.526504e+06	POLYG ((1045944.504028 215602.45239257
811	4	Queens	Bayside-Bayside Hills	146700	9465.098967	3.759538e+06	POLYG ((1048146.580993 216352.642395019
941	4	Queens	Bayside-Bayside Hills	147100	10200.518035	4.561921e+06	POLYG ((1051290.736816 216396.213989257
1513	4	Queens	Douglas Manor-Douglaston-Little Neck	147900	15085.985039	9.172315e+06	POLYG ((1057956.090393 220841.231201171
961	4	Queens	Douglas Manor-Douglaston-Little Neck	148300	17873.554044	1.472603e+07	POLYG ((1056433.537841 222106.904602050

782	3	Brooklyn	Windsor Terrace	150200	11171.749878	3.430497e+06	POLYG ((988387.68841552 179373.051208496
942	4	Queens	Douglas Manor-Douglaston-Little Neck	150701	15038.633567	9.372852e+06	POLYG ((1058514.6622314 216501.548400878
1514	4	Queens	Douglas Manor-Douglaston-Little Neck	150702	13235.675959	8.964553e+06	POLYG ((1061604.9086303 217959.83319091
1544	3	Brooklyn	Flatbush	152200	9342.803909	4.192020e+06	POLYG ((994216.12799072 174053.055236816
1326	4	Queens	Douglas Manor-Douglaston-Little Neck	152901	16343.845155	1.502207e+07	POLYG ((1060212.3621826 213475.711608886
1928	4	Queens	Douglas Manor-Douglaston-Little Neck	152902	14314.661072	1.112033e+07	POLYG ((1063614.4486083 216235.869995117
1716	4	Queens	Glen Oaks-Floral Park-New Hyde Park	155101	10130.321708	5.175924e+06	POLYG ((1065195.9544067 214921.718994140
1929	4	Queens	Glen Oaks-Floral Park-New Hyde Park	155102	23459.543800	1.554942e+07	POLYG ((1065195.9544067 214921.718994140
1481	4	Queens	Bellerose	156700	25839.737656	1.568104e+07	POLYG ((1062126.413391 211533.514038085
713	4	Queens	Bellerose	157101	16809.567999	1.338146e+07	POLYG ((1062495.2839965 210472.384399414
1471	4	Queens	Bellerose	157102	9503.313231	3.383205e+06	POLYG ((1060988.331604(205897.223022460
581	4	Queens	Glen Oaks-Floral Park-New Hyde Park	157901	13516.732642	9.833175e+06	POLYG ((1067382.5084228 208774.415588378
582	4	Queens	Glen Oaks-Floral Park-New Hyde Park	157902	11455.796897	7.773114e+06	POLYG ((1065581.8295898 208184.836791992
583	4	Queens	Glen Oaks-Floral Park-New Hyde Park	157903	12918.427452	7.335307e+06	POLYG ((1064489.8829956 204357.586791992
1167	4	Queens	Bellerose	161700	11944.490409	7.598658e+06	POLYG ((1064489.8829956 204357.586791992
1696	4	Queens	Bellerose	162100	13356.573840	7.726052e+06	POLYG ((1059728.3375854 205755.894592285
1662	5	Staten Island	park-cemetery-etc-Staten Island	990100	4743.128085	6.357020e+05	(POLYG ((970217.0223999(145643.332214355

2166 rows × 7 columns

The second step is to obtain the census data for the variables we have selected. Here come into play the various function declared before, starting from the CENSUS_DATA, which is used in the creation of a dictionary called **NYC_DATA**. This dictionary is created thorugh a conditional function which, iterating through *census_data,* sets as keys the values of the census codes used, while setting as values dictionaries containing dataframes with the data for each borough of every census variable.

In [13]:
```
NYC_DATA = {census_data.get(y):
             {x:
              pd.DataFrame(CENSUS_DATA(y, NYC_BOROUGHS.get(x)[0]))
              for x in NYC_BOROUGHS
             }
             for y in census_data
            }
```

Below we can see an example of one of the dataframes contained in the dictionary of dictionaries just created. As we can see from the data, this dataframe need to be cleaned to be usable, since it contains outliers like the -666666666.0 values, which are certainly indicating missing data, as well as 0s.

In [14]:
```
NYC_DATA['Median Household Income']['manhattan']
```

Out[14]:

	B19013_001E	county	state	tract
0	-666666666.0	061	36	000100
1	21102.0	061	36	000201
2	32411.0	061	36	000202
3	-666666666.0	061	36	000500
4	18641.0	061	36	000600
5	120291.0	061	36	000700
6	31129.0	061	36	000800
7	158125.0	061	36	000900
8	75192.0	061	36	001001
9	16464.0	061	36	001002
10	51474.0	061	36	001200
11	124313.0	061	36	001300
12	75556.0	061	36	001401
13	26971.0	061	36	001402
14	85426.0	061	36	001501
15	145387.0	061	36	001502
16	42986.0	061	36	001600
17	61750.0	061	36	001800
18	15490.0	061	36	002000
19	250001.0	061	36	002100
20	33700.0	061	36	002201
21	88099.0	061	36	002202

22	18354.0	061	36	002400
23	16324.0	061	36	002500
24	56420.0	061	36	002601
25	73056.0	061	36	002602
26	56422.0	061	36	002700
27	35440.0	061	36	002800
28	30434.0	061	36	002900
29	71591.0	061	36	003001
...	...	...	...	...
258	47650.0	061	36	025300
259	45852.0	061	36	025500
260	76832.0	061	36	025700
261	55903.0	061	36	025900
262	31960.0	061	36	026100
263	28630.0	061	36	026300
264	66217.0	061	36	026500
265	36935.0	061	36	026700
266	33073.0	061	36	026900
267	53842.0	061	36	027100
268	75180.0	061	36	027300
269	91528.0	061	36	027500
270	25896.0	061	36	027700
271	46395.0	061	36	027900
272	101477.0	061	36	028100
273	51948.0	061	36	028300
274	32164.0	061	36	028500
275	47477.0	061	36	028700
276	30878.0	061	36	029100
277	34688.0	061	36	029300
278	66127.0	061	36	029500
279	-666666666.0	061	36	029700
280	24625.0	061	36	029900
281	58190.0	061	36	030300
282	72212.0	061	36	030700
283	27675.0	061	36	030900
284	-666666666.0	061	36	031100
285	180742.0	061	36	031703
286	156984.0	061	36	031704
287	-666666666.0	061	36	031900

288 rows × 4 columns

We then proceed to format these dataframes contained in our new dictionary using the function *CENSUS_DF_FM* defined earlier.

In [15]:
```
for x in NYC_DATA:
    CENSUS_DF_FM(NYC_DATA[x], x)
```

Since we decided to study also tenure, we thought it would be more reasonable to study the percentage of owned and rented houses, rather than the exact numbers. For this reason, below we set the code to convert the columns 'Tenure: Owned' and 'Tenure: Rented' into percentage values over total tenure in the nyc_data Data Frame.

In [16]:
```
for x in NYC_DATA['Tenure: Owned']:
    NYC_DATA['Tenure: Owned'][x]['Tenure: Owned'] = NYC_DATA['Tenure: Owned'][x
][['Tenure: Owned']/NYC_DATA['Tenure: Total'][x]['Tenure: Total']
    NYC_DATA['Tenure: Rented'][x]['Tenure: Rented'] = NYC_DATA['Tenure: Rented'
][x]['Tenure: Rented']/NYC_DATA['Tenure: Total'][x]['Tenure: Total']
```

In order to be able to use this data with the Shapefiles added, we need to create a new dictionary **nyc_data_draft** with keys corresponding to the census data names and as values dataframes generate from the concatenation of the borough dataframes in NYC_DATA. This will leave us with dataframes containing the data for all boroughs combined, for each census variable.

Then we create a final DataFrame **nyc_data** containing all our variables cleaned and ordered, which displays in a readable way our data.

In [17]:
```
nyc_data_draft = {y: pd.concat([NYC_DATA[y][x] for x in NYC_DATA[y]], axis=0) f
or y in NYC_DATA}
nyc_data = nyc_data_draft[list(census_data.values())[0]]
nyc_data = nyc_data.set_index(nyc_data_draft[list(census_data.values())[0]].ind
ex)
for x in list(nyc_data_draft.keys())[:0:-1]:
    nyc_data.insert(1, x, nyc_data_draft[x][x])

for x in list(nyc_data_draft.keys()):
    nyc_data = nyc_data[nyc_data[x] != 0]
    nyc_data = nyc_data[nyc_data[x] != -666666666.0]
```

In [18]:
```
nyc_data
```

Out[18]:

	Median Household Income	Median Contract Rent	Median House Value	Tenure: Total	Tenure: Owned	Tenure: Rented	BoroName	CT2010
2	32411.0	713.0	544200.0	3541.0	0.222254	0.777746	061	000202
4	18641.0	596.0	596500.0	4161.0	0.040135	0.959865	061	000600
5	120291.0	2846.0	1370900.0	4511.0	0.198847	0.801153	061	000700
6	31129.0	934.0	545500.0	3431.0	0.010493	0.989507	061	000800
7	158125.0	3501.0	1070700.0	843.0	0.105575	0.894425	061	000900
8	75192.0	991.0	562700.0	790.0	0.718987	0.281013	061	001001
10	51474.0	822.0	546100.0	1870.0	0.365241	0.634759	061	001200
11	124313.0	2764.0	939800.0	2470.0	0.094332	0.905668	061	001300
12	75556.0	1167.0	657300.0	1676.0	0.778043	0.221957	061	001401
13	26971.0	884.0	592600.0	1378.0	0.044993	0.955007	061	001402
14	85426.0	2253.0	554600.0	3693.0	0.430003	0.569997	061	001501

15	145387.0	2996.0	1027800.0	3138.0	0.223072	0.776928	061	001502
17	61750.0	1196.0	1064300.0	3120.0	0.064744	0.935256	061	001800
19	250001.0	3501.0	2000001.0	2744.0	0.466108	0.533892	061	002100
21	88099.0	1899.0	635400.0	1077.0	0.090994	0.909006	061	002202
24	56420.0	835.0	392600.0	1583.0	0.099179	0.900821	061	002601
25	73056.0	991.0	661000.0	2118.0	0.231822	0.768178	061	002602
26	56422.0	929.0	602200.0	627.0	0.491228	0.508772	061	002700
27	35440.0	800.0	457600.0	3380.0	0.097633	0.902367	061	002800
29	71591.0	1982.0	584800.0	2322.0	0.029716	0.970284	061	003001
30	52479.0	1129.0	606500.0	1759.0	0.087550	0.912450	061	003002
31	185833.0	2752.0	1418900.0	740.0	0.286486	0.713514	061	003100
32	66835.0	1682.0	654800.0	4591.0	0.119800	0.880200	061	003200
33	207460.0	2962.0	2000001.0	2421.0	0.432879	0.567121	061	003300
34	77254.0	1984.0	711500.0	3705.0	0.066397	0.933603	061	003400
35	48990.0	960.0	435500.0	1476.0	0.097561	0.902439	061	003601
36	77778.0	1537.0	447800.0	1308.0	0.070336	0.929664	061	003602
37	145000.0	2123.0	1484000.0	1348.0	0.471810	0.528190	061	003700
38	60450.0	1701.0	852300.0	5153.0	0.097419	0.902581	061	003800
39	158646.0	1798.0	2000001.0	2572.0	0.353810	0.646190	061	003900
...	...	...	...	...	...	...	...	...
78	80357.0	1116.0	413300.0	1320.0	0.684091	0.315909	085	020100
79	31040.0	1022.0	337600.0	1825.0	0.439452	0.560548	085	020700
80	80567.0	1350.0	475100.0	3250.0	0.863385	0.136615	085	020801
81	101176.0	993.0	605100.0	2197.0	0.837961	0.162039	085	020803
82	95417.0	1064.0	602200.0	1988.0	0.909960	0.090040	085	020804
83	57659.0	996.0	328100.0	1471.0	0.584636	0.415364	085	021300
84	58523.0	950.0	329000.0	861.0	0.709640	0.290360	085	022300
85	101065.0	1187.0	504700.0	2801.0	0.797929	0.202071	085	022600
87	70776.0	1330.0	326600.0	1260.0	0.719841	0.280159	085	023100
88	60240.0	893.0	351500.0	1015.0	0.613793	0.386207	085	023900
89	75489.0	977.0	674600.0	2086.0	0.755992	0.244008	085	024401
90	83269.0	1092.0	594700.0	1552.0	0.784794	0.215206	085	024402
91	59145.0	1158.0	357000.0	981.0	0.555556	0.444444	085	024700
92	75625.0	990.0	548700.0	1678.0	0.746722	0.253278	085	024800
93	97316.0	1328.0	441500.0	2083.0	0.851176	0.148824	085	025100
94	68546.0	1226.0	385600.0	1541.0	0.791694	0.208306	085	027301
95	88958.0	1167.0	511800.0	1238.0	0.651858	0.348142	085	027302
96	58716.0	1244.0	302300.0	2977.0	0.674505	0.325495	085	027702
97	98722.0	1209.0	467200.0	1483.0	0.766689	0.233311	085	027704
98	79583.0	1146.0	472300.0	1912.0	0.775628	0.224372	085	027705
99	81654.0	1168.0	467500.0	1155.0	0.733333	0.266667	085	027706

100	91944.0	879.0	743300.0	640.0	0.818750	0.181250	085	027900
101	89167.0	1250.0	397700.0	954.0	0.839623	0.160377	085	029102
102	74732.0	898.0	481800.0	2602.0	0.760184	0.239816	085	029103
103	87853.0	1178.0	399300.0	2383.0	0.812002	0.187998	085	029104
104	65469.0	1162.0	333400.0	1817.0	0.642818	0.357182	085	030301
105	84630.0	1448.0	322000.0	1962.0	0.757900	0.242100	085	030302
106	23098.0	427.0	294700.0	1038.0	0.259152	0.740848	085	031901
107	23289.0	735.0	281100.0	1588.0	0.418136	0.581864	085	031902
108	64250.0	1083.0	231900.0	499.0	0.691383	0.308617	085	032300

1913 rows × 8 columns

Similarly to what we have done with the census data, we now create a new dictionary of dictionaries called **NYC_SHP**, which we format using the **SHP_FM** function create before, and finally create a dictionary of DataFrames **nyc_shp**, which will be the one we will use to plot the data through the choropleth maps.

In [19]:
```
NYC_SHP = {census_data.get(y):
           {x:
            NY_shp.loc[NY_shp['BoroCode']==NYC_BOROUGHS.get(x)[1]] for x in NYC
_BOROUGHS
           }
           for y in census_data
          }

SHP_FM(NYC_SHP)

nyc_shp = {y: pd.concat([NYC_SHP[y][x] for x in NYC_SHP[y]], axis=0) for y in N
YC_SHP}
```

In [20]:
```
nyc_shp['Median Household Income']
```

Out[20]:

	BoroCode	BoroName	NTAName	Median Household Income	CT2010	Shape_Leng	Shape_Area	
1	1	Manhattan	Lower East Side	21102.0	000201	4748.325960	9.715994e+05	((98854 197770.
2	1	Manhattan	Lower East Side	32411.0	000202	8568.089235	3.315114e+06	((98913 196325
4	1	Manhattan	Lower East Side	18641.0	000600	6970.634699	2.583418e+06	((98696 199553.
5	1	Manhattan	Battery Park City-Lower Manhattan	120291.0	000700	9802.917997	2.442012e+06	((98480 193867
6	1	Manhattan	Chinatown	31129.0	000800	6362.785762	2.366871e+06	((98649 197775.
7	1	Manhattan	Battery Park City-Lower Manhattan	158125.0	000900	16484.098901	3.122059e+06	((9840 192223.

8	1	Manhattan	Lower East Side	75192.0	001001	3924.786593	8.678059e+05	((99112 199590
9	1	Manhattan	Lower East Side	16464.0	001002	6315.122079	2.225720e+06	((99159 201146.
10	1	Manhattan	Lower East Side	51474.0	001200	5462.924251	1.298211e+06	((98963 200301.
11	1	Manhattan	Battery Park City-Lower Manhattan	124313.0	001300	8906.314403	3.516583e+06	((98186 198455.
12	1	Manhattan	Lower East Side	75556.0	001401	5075.332269	1.006117e+06	((9874 20029
13	1	Manhattan	Lower East Side	26971.0	001402	4459.156254	1.226206e+06	((98838 201258.
14	1	Manhattan	Battery Park City-Lower Manhattan	85426.0	001501	9130.652444	2.814944e+06	((98563 195452
15	1	Manhattan	Battery Park City-Lower Manhattan	145387.0	001502	10812.023697	2.242453e+06	((98552 19526
16	1	Manhattan	Chinatown	42986.0	001600	6391.788333	2.233320e+06	((98725 200392.
17	1	Manhattan	Chinatown	61750.0	001800	6391.922373	2.399268e+06	((9870 20178
18	1	Manhattan	Lower East Side	15490.0	002000	4690.072720	1.368856e+06	((99176 201989.
19	1	Manhattan	SoHo-TriBeCa-Civic Center-Little Italy	250001.0	002100	6824.353555	2.618006e+06	((98169 200203.
20	1	Manhattan	Lower East Side	33700.0	002201	5779.062466	1.740174e+06	((99013 201568.
21	1	Manhattan	Lower East Side	88099.0	002202	3817.391260	6.039222e+05	((99028 201838.
22	1	Manhattan	Lower East Side	18354.0	002400	8143.066370	1.860513e+06	((99335 200980
23	1	Manhattan	Chinatown	16324.0	002500	4763.702704	1.439388e+06	((98535 197553.
24	1	Manhattan	Lower East Side	56420.0	002601	4521.250996	1.141217e+06	((9906 20250
25	1	Manhattan	Lower East Side	73056.0	002602	4491.203470	1.114857e+06	((99101

								203153.
26	1	Manhattan	Chinatown	56422.0	002700	3685.315982	6.392992e+05	((98509 198703
27	1	Manhattan	Lower East Side	35440.0	002800	5627.585497	1.973554e+06	((99165 204299.
28	1	Manhattan	Chinatown	30434.0	002900	8978.442335	3.010944e+06	((98534 200219.
29	1	Manhattan	Chinatown	71591.0	003001	4758.460807	1.344751e+06	((98873 202128.
30	1	Manhattan	East Village	52479.0	003002	4045.166292	8.410978e+05	((98897 202565.
31	1	Manhattan	SoHo-TriBeCa-Civic Center-Little Italy	185833.0	003100	7779.903923	2.208672e+06	((98426 200869.
...	...	...	...	...	...	...	...	
78	5	Staten Island	Westerleigh	80357.0	020100	14496.555140	6.756463e+06	((94803 169151.
79	5	Staten Island	Port Richmond	31040.0	020700	21681.724166	9.979093e+06	((94924 172931
80	5	Staten Island	Rossville-Woodrow	80567.0	020801	20661.370367	2.801678e+07	((9277 14193
81	5	Staten Island	Rossville-Woodrow	101176.0	020803	19428.715667	2.262199e+07	((92832 131251.
82	5	Staten Island	Rossville-Woodrow	95417.0	020804	16137.209945	1.432458e+07	((93038 134554.
83	5	Staten Island	Port Richmond	57659.0	021300	14576.898641	8.433154e+06	((94803 169440.
84	5	Staten Island	Mariner's Harbor-Arlington-Port Ivory-Granitev...	58523.0	022300	24371.902096	7.143291e+06	((94452 172361
85	5	Staten Island	Charleston-Richmond Valley-Tottenville	101065.0	022600	54955.968358	8.623355e+07	((92673 142481.
87	5	Staten Island	Mariner's Harbor-Arlington-Port Ivory-Granitev...	70776.0	023100	10150.553578	6.280175e+06	((94174 167235.
88	5	Staten Island	Mariner's Harbor-Arlington-Port Ivory-Granitev...	60240.0	023900	10994.801543	5.676613e+06	((94347 169257.

89	5	Staten Island	Charleston-Richmond Valley-Tottenville	75489.0	024401	24555.316105	2.256075e+07	((9212 12850
90	5	Staten Island	Charleston-Richmond Valley-Tottenville	83269.0	024402	22532.187608	1.927884e+07	((92255 122443.
91	5	Staten Island	Port Richmond	59145.0	024700	10771.844611	5.207304e+06	((94629 168344
92	5	Staten Island	Charleston-Richmond Valley-Tottenville	75625.0	024800	31780.060846	1.760147e+07	((92044 128476.
93	5	Staten Island	Port Richmond	97316.0	025100	16988.928550	1.280149e+07	((94586 166735.
94	5	Staten Island	New Springville-Bloomfield-Travis	68546.0	027301	23034.313574	3.274167e+07	((94337 161026
95	5	Staten Island	New Springville-Bloomfield-Travis	88958.0	027302	15349.525180	1.100867e+07	((94787 157718.
96	5	Staten Island	Todt Hill-Emerson Hill-Heartland Village-Light...	58716.0	027702	17816.578948	1.570069e+07	((93897 152923.
97	5	Staten Island	Todt Hill-Emerson Hill-Heartland Village-Light...	98722.0	027704	10021.995797	5.668295e+06	((94374 153739.
98	5	Staten Island	Todt Hill-Emerson Hill-Heartland Village-Light...	79583.0	027705	9896.267587	5.721087e+06	((9423 15433
99	5	Staten Island	Todt Hill-Emerson Hill-Heartland Village-Light...	81654.0	027706	9395.468335	4.718053e+06	((94082 155432.
100	5	Staten Island	Todt Hill-Emerson Hill-Heartland Village-Light...	91944.0	027900	36228.516575	5.835092e+07	((94895 149302.
101	5	Staten Island	New Springville-Bloomfield-Travis	89167.0	029102	58467.453836	1.064312e+08	((92997 170693.
102	5	Staten Island	New Springville-Bloomfield-Travis	74732.0	029103	25335.750447	2.197857e+07	((93915 160533.

103	5	Staten Island	New Springville-Bloomfield-Travis	87853.0	029104	19502.863316	1.634464e+07	((94155 162375
104	5	Staten Island	Mariner's Harbor-Arlington-Port Ivory-Granitev...	65469.0	030301	18499.152245	7.812087e+06	((94365 166987.
105	5	Staten Island	Mariner's Harbor-Arlington-Port Ivory-Granitev...	84630.0	030302	13923.141469	9.788180e+06	((93969 167578.
106	5	Staten Island	Mariner's Harbor-Arlington-Port Ivory-Granitev...	23098.0	031901	8255.026898	4.126596e+06	((93969 167578.
107	5	Staten Island	Mariner's Harbor-Arlington-Port Ivory-Granitev...	23289.0	031902	18067.517433	7.019455e+06	((93978 171881.
108	5	Staten Island	Mariner's Harbor-Arlington-Port Ivory-Granitev...	64250.0	032300	37831.963735	4.120267e+07	((93674 173125.

2098 rows × 8 columns

Data Analysis

Now that the data is ready to be analyze, we can start to look at each census variable individually and then study their correlation in order to see if our variables retain a certain degree of relation in the counties of New York City.

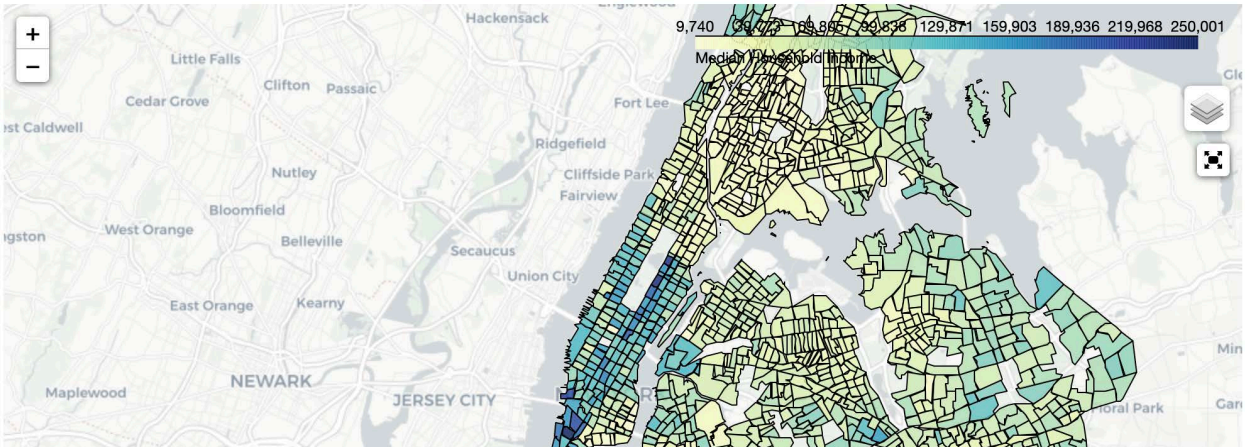
Median Household Income

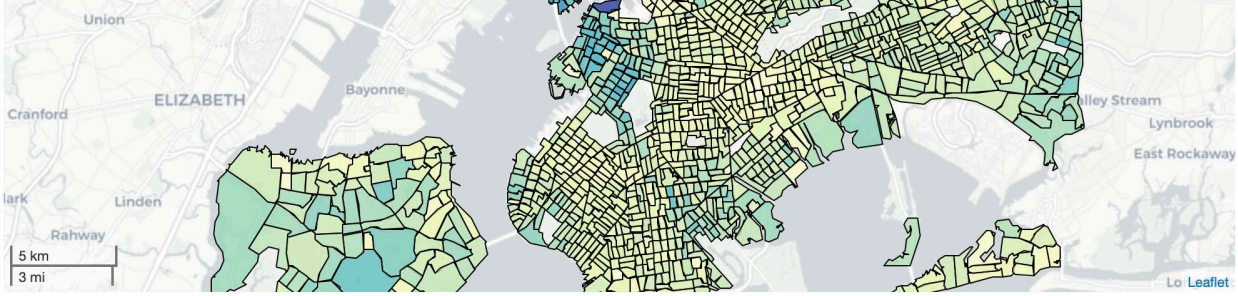
The first variable we want to look at is Median House Income for all the census tracts of the 5 boroughs.

To have a better understanding of the distribution of this variable on the territory, we can plot the choropleth map using the function *MAP* defined in the biginning.

In [21]:
```
MAP('Median Household Income', colorstyles[15])
```

Out[21]:





From what we can see from the choropleth map created, New York City counties exhibit a great household income disparity, with Manhattan showing a significant difference compared to the others. Looking more into Manhattan's income distribution, we can see how the city presents a relatively high level of income, while still retaining a strong divarious between few neighbourhoods. Downtown and Upper East Side are the areas in which we can better see this income spread: the Financial District and Tribeca, as well as the Upper East Side blocks facing Central Park, are characterized by a median household income of more than 200,000$, compared to China Town and East Harlem with incomes around 30,000$.

In [22]:
```
nyc_shp['Median Household Income'].sort_values('Median Household Income', ascending=False)[:10]
```

Out[22]:

	BoroCode	BoroName	NTAName	Median Household Income	CT2010	Shape_Leng	Shape_Area	
19	1	Manhattan	SoHo-TriBeCa-Civic Center-Little Italy	250001.0	002100	6824.353555	2.618006e+06	((98169 200203.2
141	1	Manhattan	Upper East Side-Carnegie Hill	235083.0	014200	5821.704938	1.927688e+06	((99583 223039.7
165	1	Manhattan	Upper East Side-Carnegie Hill	233529.0	016001	4702.173848	1.352632e+06	((9973 225841
12	3	Brooklyn	DUMBO-Vinegar Hill-Downtown Brooklyn-Boerum Hill	211071.0	002100	12735.721735	5.707252e+06	((99005 196474.8
67	1	Manhattan	West Village	209167.0	006900	7535.325615	2.350754e+06	((98241 205843.5
33	1	Manhattan	SoHo-TriBeCa-Civic Center-Little Italy	207460.0	003300	8170.756112	3.677727e+06	((98277 202256.5
93	1	Manhattan	Midtown-Midtown South	199167.0	009400	5738.939752	1.646379e+06	((99133 214911.5
130	1	Manhattan	Upper East Side-Carnegie Hill	191607.0	013000	5807.972956	1.918145e+06	((99492 221386.2
111	1	Manhattan	Midtown-	187500.0	011202	3688.762080	8.395256e+05	

			Midtown South					((992 21722
31	1	Manhattan	SoHo-TriBeCa-Civic Center-Little Italy	185833.0	003100	7779.903923	2.208672e+06	((984262 200869.5

Looking at the first 10 blocks with highest median household income, we also note that DUMBO is the only neighbourhood not located in the Manhattan borough to result in the list. Looking at the map again, we see the significant difference between DUMBO and the neighbouring blocks, which, even presenting incomes hovering above 100,000$, are way lower than DUMBO's 210,000$ median income.

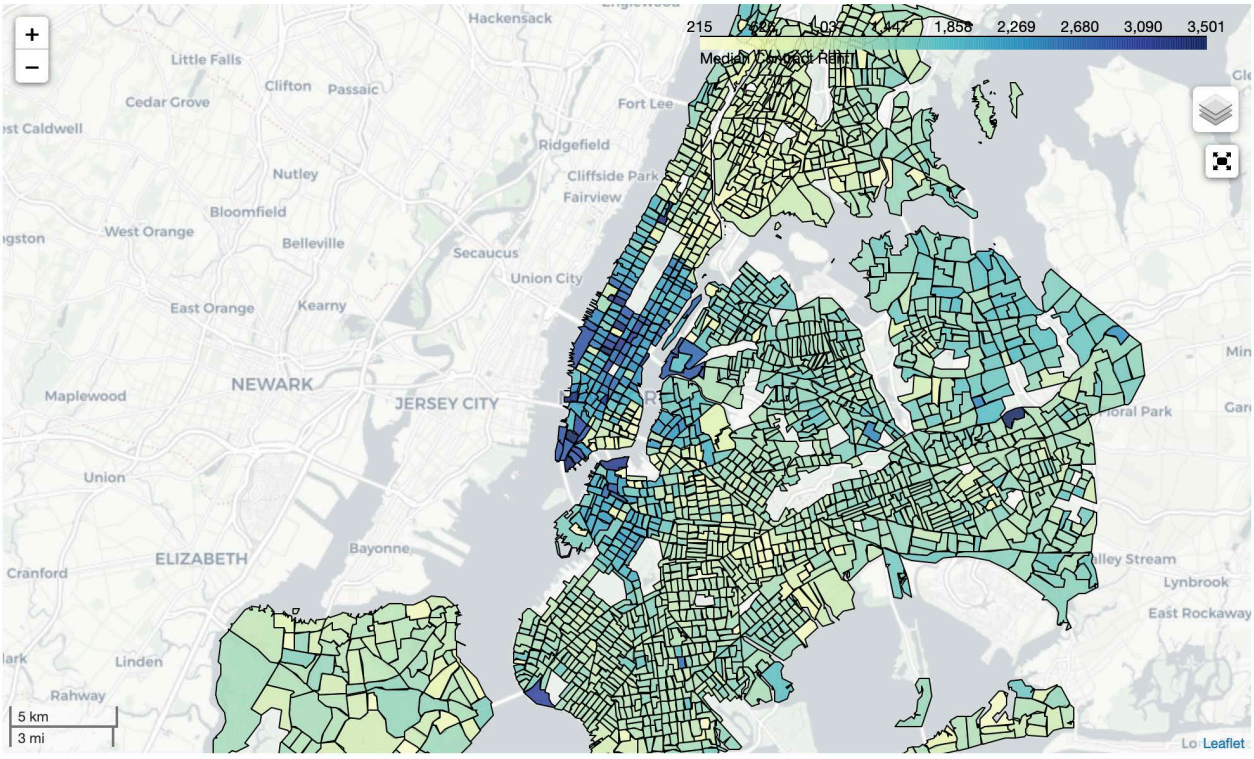
Median Contract Rent

Median Contract Rent is the second variable that we decided to analyze.

Looking at the map, we can see how in this case the data seems to be still not quite homogeneous across the five boroughs. We can still clearly see that Manhattan represents the borough with the highest rents out of all the New York City boroughs.

In [23]:
```
MAP('Median Contract Rent', colorstyles[15])
```

Out[23]:



By getting a closer look at the blocks with the highest contract rent, we can see that the majority of the blocks are located in the heart of Manhattan, mainly oround the most popular locations like Times Square, Midtown and Downtown. Furthermore, we can also note that the Queens borough is represented in the top ten highest contract rents by Jamaica Estates-Holliswood. Given the value and the position in the map, this observation may represent an outlier.

In [24]:
```
nyc_shp['Median Contract Rent'].sort_values('Median Contract Rent', ascending=False)[:10]
```

Out[24]:

	BoroCode	BoroName	NTAName	Median Contract Rent	CT2010	Shape_Leng	Shape_Area

7	1	Manhattan	Battery Park City-Lower Manhattan	3501.0	000900	16484.098901	3.122059e+06	((98403 192223.7
312	4	Queens	Jamaica Estates-Holliswood	3501.0	047600	11595.617217	5.807729e+06	((104943 203383.7
19	1	Manhattan	SoHo-TriBeCa-Civic Center-Little Italy	3501.0	002100	6824.353555	2.618006e+06	((981698 200203.2
100	1	Manhattan	Midtown-Midtown South	3444.0	010100	5701.715857	1.891378e+06	((987638 212444.5
67	1	Manhattan	West Village	3346.0	006900	7535.325615	2.350754e+06	((982414 205843.5
203	1	Manhattan	Morningside Heights	3345.0	019701	10250.870237	2.537860e+06	((996716 234408.6
125	1	Manhattan	Midtown-Midtown South	3333.0	012500	5683.937071	1.875482e+06	((989672 216129.0
83	1	Manhattan	Turtle Bay-East Midtown	3323.0	008601	6106.667391	1.216787e+06	((994988 209035.2
145	1	Manhattan	Lincoln Square	3313.0	014500	5836.620068	1.898853e+06	((989633 219774.6
82	1	Manhattan	Midtown-Midtown South	3191.0	008400	5690.701888	1.862742e+06	((989545 213790.7

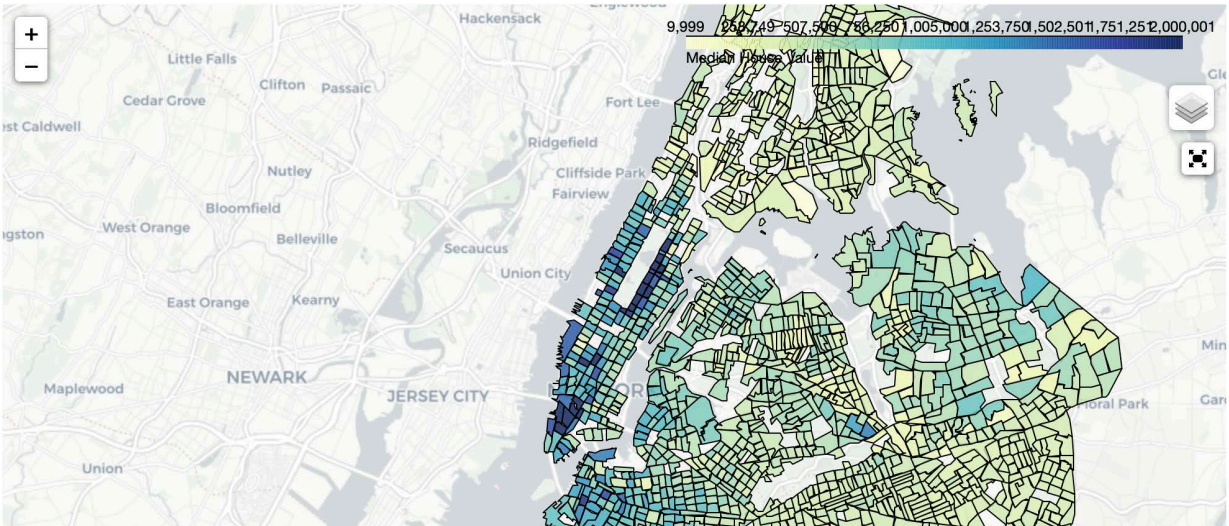
Median House Value

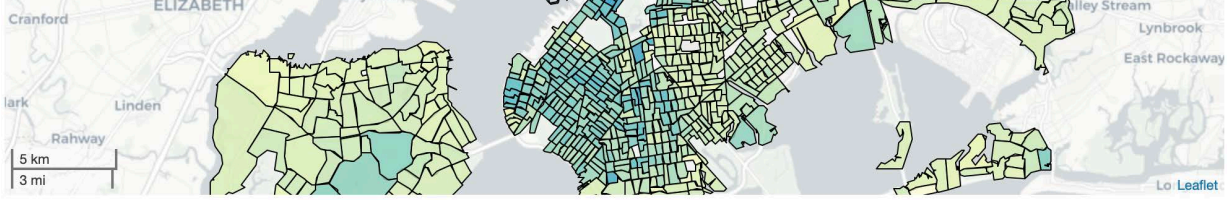
In order to study the New York City Housing Market, we decided to use the Median House Values provided by the Census.

From what we can see, the map presents a higher number of missing values. This may be related to the fact that this data usually refers to house selling quotes, which sometimes may not be available. Looking at the map, we can see how in this case the data seems to be still not quite homogeneous across the five boroughs. We can still clearly see that Manhattan represents the borough with the highest rents out of all the New York City boroughs.

In [25]:
```
MAP('Median House Value', colorstyles[15])
```

Out[25]:





Looking at the census blocks with the highest Median House Value, we see how the data has been certainly manipulated to fit an upper bound. This unfortunately may not benefit to our goal of fully analyzing the census data. In any case, we can still come to the conclusion that Manhattan reigns in terms of house values in New York City, since, as we can see, all the 10 highest median house values are belonging to blocks located in Manhattan.

In [26]:
```
nyc_shp['Median House Value'].sort_values('Median House Value', ascending=False)[:10]
```

Out[26]:

	BoroCode	BoroName	NTAName	Median House Value	CT2010	Shape_Leng	Shape_Area	
128	1	Manhattan	Upper East Side-Carnegie Hill	2000001.0	012800	5818.320510	1.924403e+06	((995812.3 220891.66
174	1	Manhattan	East Harlem South	2000001.0	016800	6262.354696	2.151828e+06	((998675.7 228167.94
33	1	Manhattan	SoHo-TriBeCa-Civic Center-Little Italy	2000001.0	003300	8170.756112	3.677727e+06	((982778.1 202256.59
123	1	Manhattan	Upper East Side-Carnegie Hill	2000001.0	012200	5683.168073	1.857830e+06	((994005.1 219736.09
153	1	Manhattan	Upper East Side-Carnegie Hill	2000001.0	015002	4689.906442	1.352304e+06	((996746.1 224680.32
86	1	Manhattan	Hudson Yards-Chelsea-Flatiron-Union Square	2000001.0	008700	5579.480631	1.780834e+06	((986109.4 209692.65
150	1	Manhattan	Upper East Side-Carnegie Hill	2000001.0	014802	4691.030684	1.351073e+06	((997637.4 224187.03
45	1	Manhattan	SoHo-TriBeCa-Civic Center-Little Italy	2000001.0	004500	4604.742526	1.129524e+06	((985047.5 202430.19
19	1	Manhattan	SoHo-TriBeCa-Civic Center-Little Italy	2000001.0	002100	6824.353555	2.618006e+06	((981698.1 200203.23
130	1	Manhattan	Upper East Side-	2000001.0	013000	5807.972956	1.918145e+06	((994920.1

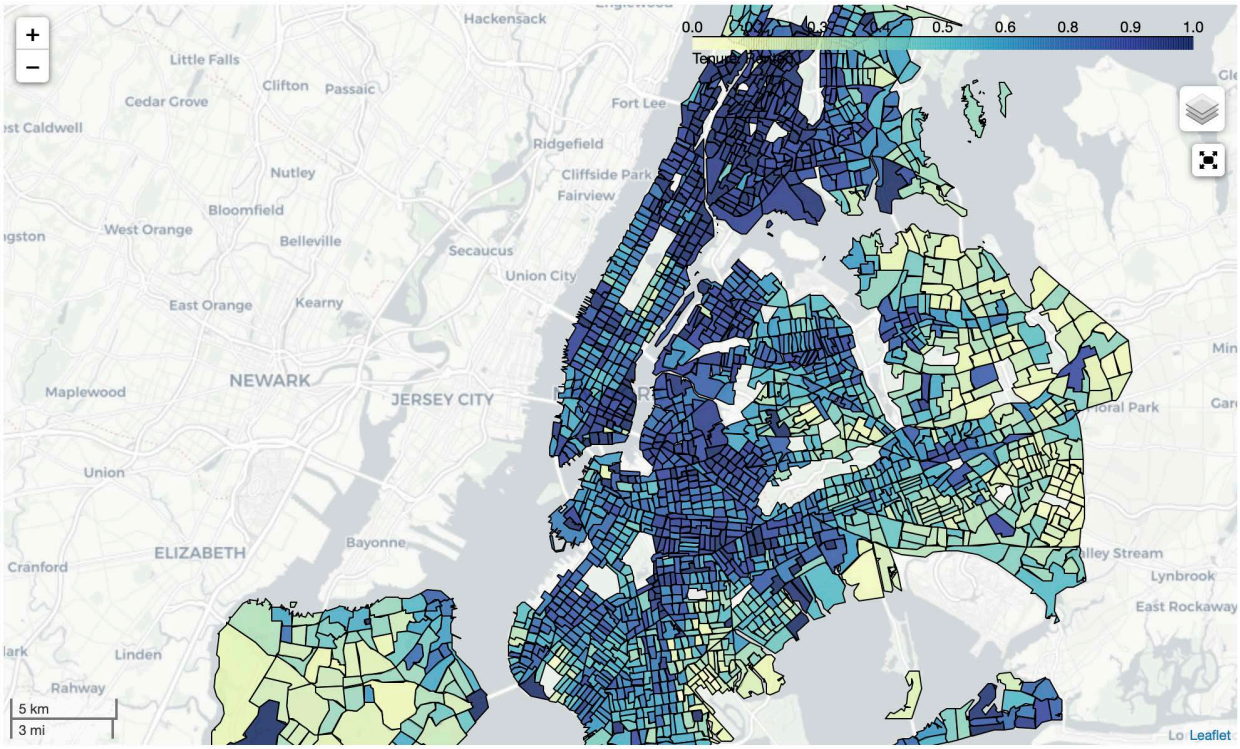
	Carnegie Hill	221386.26

Tenure

We chose tenure as last variable which may be interesting to look at for the purpose of understanding the characteristics of New York City's housing market. We picked both the number of houses owned and rented, in order to be able to compare the percentage of houses either fully onwed or under a rental contract.

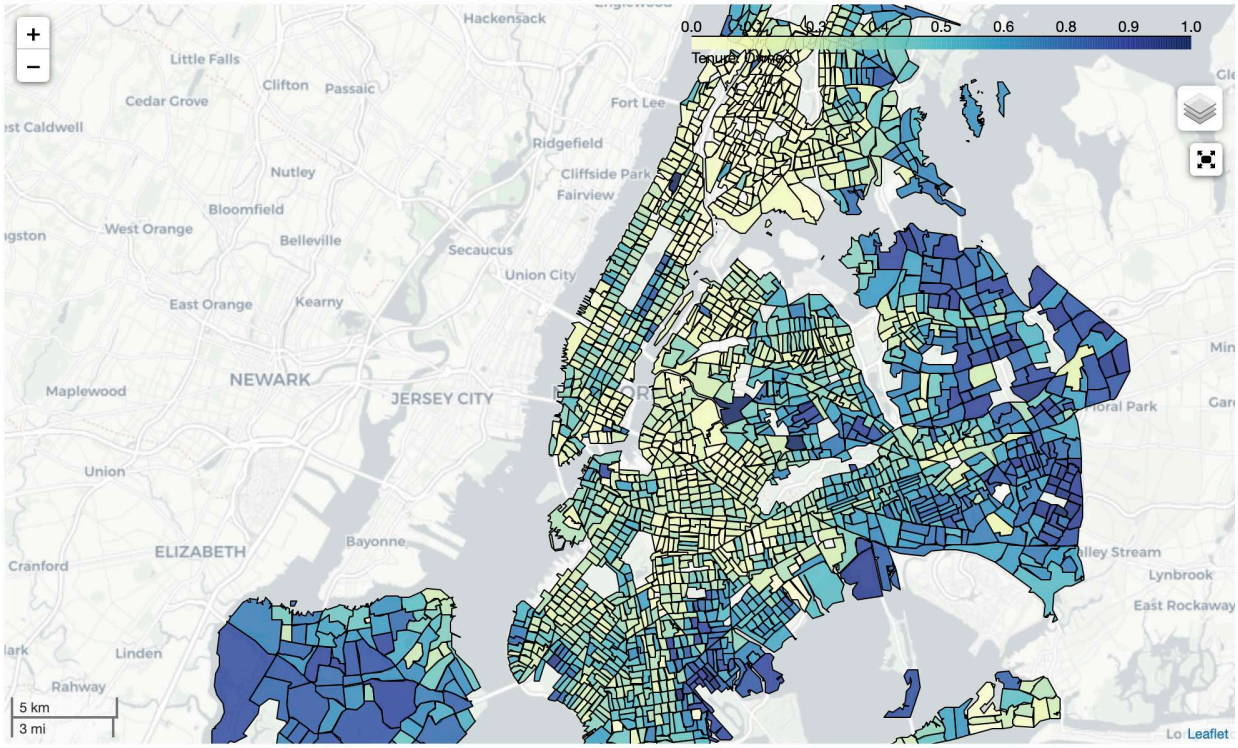
In [27]:
```
MAP('Tenure: Rented', colorstyles[15])
```

Out[27]:



In [28]:
```
MAP('Tenure: Owned', colorstyles[15])
```

Out[28]:



As we can se from the two maps, the percentage of houses owned is highest in Queens and Staten Island, while Manhattan shows a more strong ratio of rented properties.

However, the Upper East Side facing Central Park seems to be characterised by a higher number of owned estates, which may be related to the fact that the area has more higher income residents that may be interested in investing in real estate.

In [29]:
```
nyc_shp['Tenure: Owned'].sort_values('Tenure: Owned', ascending=False)[:10]
```

Out[29]:

	BoroCode	BoroName	NTAName	Tenure: Owned	CT2010	Shape_Leng	Shape_Area	
410	4	Queens	Ridgewood	1.000000	061302	10370.460842	5.511491e+06	((1017 196679
164	4	Queens	Hunters Point-Sunnyside-West Maspeth	1.000000	021900	22449.311164	1.375443e+07	((1009 204310
223	1	Manhattan	Manhattanville	1.000000	021703	9422.114991	3.384834e+06	((9991 238490
416	4	Queens	Laurelton	0.982490	062000	6540.979465	2.387806e+06	((1057 188022
533	4	Queens	Breezy Point-Belle Harbor-Rockaway Park-Broad ...	0.969920	091601	48385.575061	4.702112e+07	((1016 146374
535	3	Brooklyn	Georgetown-Marine Park-Bergen Beach-Mill Basin	0.969697	065400	6780.003081	1.631232e+06	((1004 162111
483	3	Brooklyn	Madison	0.968696	056200	5590.084972	1.838821e+06	((9993 159610
538	3	Brooklyn	Georgetown-Marine Park-Bergen Beach-Mill Basin	0.953488	066000	5223.596856	1.639605e+06	((1004 160582
408	4	Queens	Cambria Heights	0.951076	061200	7123.412870	2.830128e+06	((1058 190483
539	4	Queens	Breezy Point-Belle Harbor-Rockaway Park-Broad ...	0.942549	092800	12973.832347	8.206267e+06	((1026 147970

From what we can see, the data seems to have been modified by the Census by setting a threshold for each variable, given the fact that the values analyzed all present an upper bound after a certain number.

This may work against us the moment we would like to analyze the correlation of our variables, which is the next step we will pursue in our data analysis.

Taking into account of such problematic, we now move on to plot the correlation matrix of the census variables chosen and analyze it to see if we can find any interesting pattern to further study.

We start by calculating the correlation matrix table for all the variables.

As we can see from the table, we have a strong positive correlation between Median Household Income and Median Contract Rent, while Median House Value and the first two show a moderate positive correlation around 0.5. Also the percentage of owned properties shows a positive correlation with Median Household income, while the percentage of Rented Homes obviously has a negative correlation opposite to the one shown by Tenure: Owned. Median Contract Rent and ratio of owned homes shows a quite weak relation, while Median House value has a weak negative corelation with the first variable.

In [30]:
```
cmatrix = nyc_data.corr()
cmatrix
```

Out[30]:

	Median Household Income	Median Contract Rent	Median House Value	Tenure: Total	Tenure: Owned	Tenure: Rented
Median Household Income	1.000000	0.777560	0.565263	0.138974	0.447121	-0.447121
Median Contract Rent	0.777560	1.000000	0.546270	0.184925	0.132791	-0.132791
Median House Value	0.565263	0.546270	1.000000	0.122841	-0.062239	0.062239
Tenure: Total	0.138974	0.184925	0.122841	1.000000	-0.267481	0.267481
Tenure: Owned	0.447121	0.132791	-0.062239	-0.267481	1.000000	-1.000000
Tenure: Rented	-0.447121	-0.132791	0.062239	0.267481	-1.000000	1.000000

Since looking at numbers may not be as easy as looking at a graphical visualisation, we decided to plot a correlation matrix heatmap to better understand the strength of the correlation between these variables.

In [31]:
```
cmhp = plt.figure(figsize=(10,8))
cmhp = sns.heatmap(cmatrix, xticklabels=cmatrix.columns.values, yticklabels=cma
trix.columns.values, cmap="YlGnBu")
```

From this results stands out the relation of Rented and Owned Houses with the Median Household Income. To have a better understanding of these relationships, we could draw the scatterplot of both and analyze the traits identifiable from the plots.

In [32]:
```
fig1 = plt.figure(figsize=(16,8))
fig1 = sns.scatterplot(x='Median Household Income', y='Tenure: Rented', style=
'BoroName', data=nyc_data)
```

Starting with the relation between Median Household Income and Tenure: Rented, we can see from the plot that the correlation is moderately negative. This result tells us that Median Household Income the population tends to opt more for Owned estates rather than rented properties, which may be explained by the tendency of people to prefer investing in a fully owned home rather than

having to pay a rent when income allows it. Through this visual representation we see also the distribution of outliers in the data, which seems to be drifting the correlation to a weaker relationship. However, the correlation is still easily readable and eliminating the autliers may look unnecessary in this case.

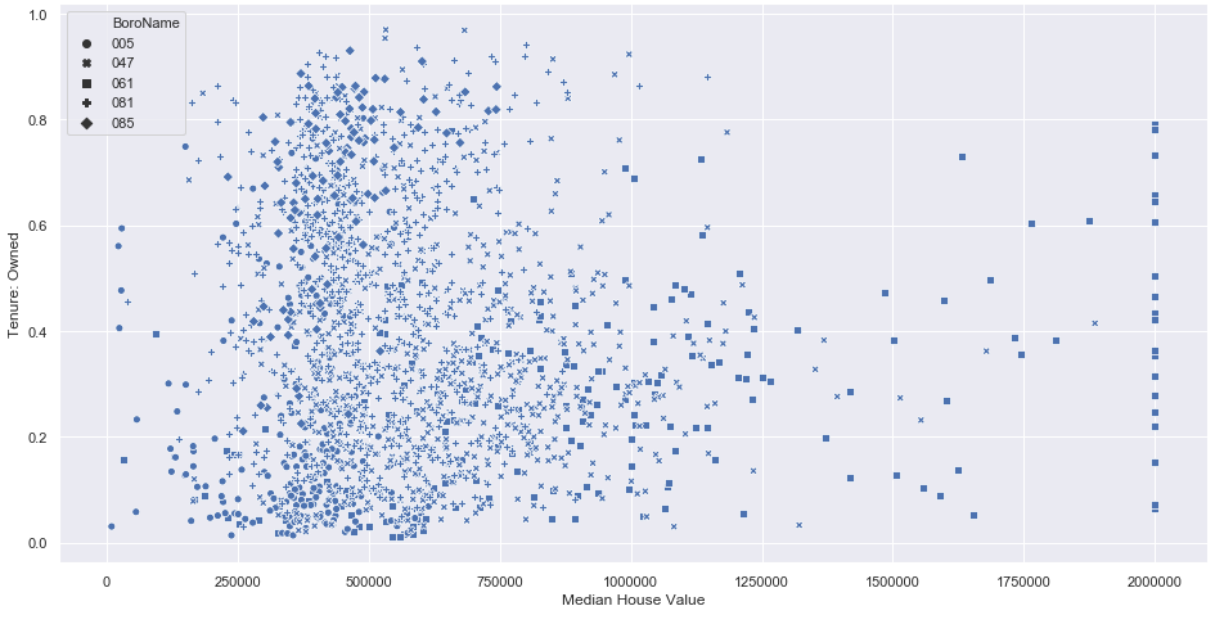
In [33]:
```
fig1 = plt.figure(figsize=(16,8))
fig1 = sns.scatterplot(x='Median Household Income', y='Tenure: Owned', style='BoroName', data=nyc_data)
```

As we would expect, when looking at the scatterplot of Median Household Income and Tenure: Owned, it look in absolute term identical to the one just plotted, with the difference that in this case we are looking at a moderate positive correlation between the two variable. As stated before, we can see that full acquisition of the properties increases as income rises.

Furthermore, we can also observe the different patterns within each borough. From the plot it looks as if there is a higher propensity to acquire a house in boroughs with relatively lower Median House Value.

To test this hypothesis, we may want to plot the scatterplot of Tenure against House Value.

In [34]:
```
fig1 = plt.figure(figsize=(16,8))
fig1 = sns.scatterplot(x='Median House Value', y='Tenure: Owned', style='BoroName', data=nyc_data)
```



What we observe here is however to partially contradict our previous statement, since we can see how the relationship is extremely weak and close to 0.

A more precise result may be drawn if we looked into more detail each borough. This would have surely helped to better understand the correlation within each New York City borough.

Conclusion

From this analysis we have been able to study the distribution of income and housing drawn from the data provided by the United States Census.

Through the use of grapichal visualization and statistics we have tried to describe how Median Household Income, Median Contract Rent, Median House Value and Tenure are spread across the five boroughs of New York City.

Our analysis has helped us to come to the conclusion that there exists a great income divarious between Manhattan and the other boroughs, which also explains the presence of relatively higher Housing Prices and Rents in the City, compared to other areas of New York City.

Moreover, we have also been able to take a closer look at the distribution of wealth within the Big Apple, identifying the richest raighbourhoods in Lower West Manhattan, The Financial District and Upper East Side, as well as DUMBO in Brooklyn. These last two Manhattan areas have also the peculiarity of a huge income divarious with neighbouring territories like Chinatown and Harlem.

Furthermore, we have been able to conclude how Income and Rent are highly correlated, while we saw that House Value and Income seem to have a weaker but still firm relationship. The correlation analysis undergone has also driven us to the conclusion that the population is more propense to acquire a house as household income rises rather than sticking with the burden of a property rent.

The code written for our analysis has been crated with the final purpose of having the possibility of visually analyzing any kind of data on New York City provided by the US Census.

The use of the census data, however, presents the drawback of using data which has already been manipulated and so may drive the user to a biased analysis.

Our project has also the potential for a deeper analysis of the New York City territory, focusing more on singular boroughs rather than a greater picture as we did. Such step may also draw more precise conclusions about the income and housing characteristics of each borough.

Data sources

https://www1.nyc.gov/site/planning/data-maps/open-data/districts-download-metadata.page

https://api.census.gov/data/2016/acs/acs5/groups.html

In [ ]: