“Blockbusting” in the 21st Century?: Minority Move-ins and Neighborhood Home Value Appreciation

Emerson Schryver

April 6, 2025

1 Project One

1.1 Introduction

Housing discrimination has, in one way or another, existed in the United States since independence. After reconstruction, several tactics became commonplace, including racial deed covenants, and redlining. Across the nation, there was evidence of “white flight’ ’, or the movement of whites out of neighborhoods with minorities due to fears over home value depreciation or other factors. Real-estate agents commonly attempted to abuse these fears with a tactic known as “blockbusting” in which they would spread fear over minority-move-ins leading to a fire sale of homes in a neighborhood (Rothstein, 2017).

Much has been researched about the modern-day effects of these past tactics, whether in deed covenants leading to improved relative neighborhood quality (Sood, Ehrman-Solberg, 2024), or redlining leading to localized areas of higher poverty (Appel, Nickerson, 2016), or simply lower quality of opportunity in majority-black neighborhoods (Chetty, et al, 2014). There is also evidence of modern-day tactics still occurring in the housing market, whether in lending markets (Quillian, Lee, Honoré, 2020), or in real estates continuing to practice “racial steering”, the process of, whether knowing it or not, primarily showing people of minority groups neighborhoods that are also primarily of that minority group (Glenn, 2018).

This paper seeks to contribute literature surrounding economic effects of racism by analyzing the modern accuracy of the perceived link that 50’s era blockbusting relied on – do minority move-ins suppress home-value appreciation?

There is a long theoretical literature on discrimination, intiially starting with a model for taste-based discrimination by Gary Becker in 1957. There now many other models for discrimination, including most famously Ken Arrow’s 1973 ‘statistical discrimininination’, but Becker’s remains the most commonly used. There are also many theoretical frameworks for home sale matching, (Badarinza, Balasubramaniam, Ramadorai, 2024), and discriminiation in labor market matching (Combes et al, 2016), but there have been few attempts to discrimination theory in the housing market. The sole notable attempt was in (Combs, et al, 2015), which built a theoretical framework for racism in home sales and lease arrangements, and then empirically tested the lease framework. This has been extended to look at a case in Moscow, where much of the racial discirmination is overt (Avetian, 2022)

This paper contributes to this literature by applying this theoretical framework to home sales, and, in particular, analyzing the long-term effects on home values. I use loan data from Fannie Mae (FNMA) & Freddie Mac (FHLMC) to provide data on neighborhood move-ins, American

Community Survey data to normalize, and Zillow ZHVI data to show the change in home prices. My study has the following methodology:

First, I select zip codes with move-ins during the 2009-2010 period (Initial Period) that are majority white (we are uninterested in the effect on majority-minority neighborhoods), and in metropolitan statistical areas (necessary for appreciation normalization). I take the "minority move-in share", or the ratio of loans for new purchases made in that zip code to minorities, and then look at those same zip codes over the 2012-2013 period (Treatment Period) and analyze what "treatment" was applied (the difference in minority move-in share from the previous period), which is our main explanatory variable. Next, we analyze the association between the change in minority move-in-share and the future appreciation in home prices, relative to their MSA average, for the next 6 years (ending in 2019). This should inform us what the effect of a sudden increase in minority move-ins is on home value appreciation.

1.2 Data Loading

1.2.1 Loading Loan Data

[2]:
```python
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
```

[3]:
```python
def read_columns_from_txt(filename):
    data = {}
    with open(filename, "r") as f:
        for line in f:
            key, value = line.split("=", 1)
            data[key.strip()] = eval(value.strip())
    return data

columns_data = read_columns_from_txt("data/columns.txt")
cols = columns_data["cols"]
cols_pre2018 = columns_data["cols_pre2018"]
dropcols = columns_data["dropcols"]
dropcols_pre2018 = columns_data["dropcols_pre2018"]
cols_2013 = columns_data["cols_2013"]
cols_2009 = columns_data["cols_2009"]
dropcols_2009 = columns_data["dropcols_2009"]
```

[4]:
```python
def load_loans(years):
    loans_list = []
    for year in years:
        if year >= 2018: # The data format changes in 2018
            files = [f"data/sf/fhlmc_sf{year}c_loans.txt", f"data/sf/
↪fnma_sf{year}c_loans.txt"]
            loans_year = pd.concat(
```

```
                [pd.read_csv(file, sep=r"\s+", header=None, names=cols).
↪drop(columns=dropcols) for file in files],
                ignore_index=True
            )
        elif year in range(2010, 2018):
            files = [f"data/sf/fhlmc_sf{year}c_loans.txt", f"data/sf/
↪fnma_sf{year}c_loans.txt"]
            loans_year = pd.concat(
                [pd.read_csv(file, sep=r"\s+", header=None, names=cols_2013).
↪drop(columns=dropcols_pre2018)  for file in files],
                ignore_index=True
            )
        elif year in range(2009, 2010):
            files = [f"data/sf/fhlmc_sf{year}c_loans.txt", f"data/sf/
↪fnma_sf{year}c_loans.txt"]
            loans_year = pd.concat(
                [pd.read_csv(file, sep=r"\s+", header=None, names=cols_2009).
↪drop(columns=dropcols_2009)  for file in files],
                ignore_index=True
            )
        loans_year["year"] = year
        loans_list.append(loans_year)
        print(f"Processed {year}", end=" ")
    return pd.concat(loans_list, ignore_index=True)
```

[5]:
```
loans = load_loans([2009, 2010, 2012, 2013]) # Please forgive the print␣
↪statement! The full data can take upwards of 20 mins to load.
```

```
Processed 2009 Processed 2010 Processed 2012 Processed 2013
```

I picked 2009 and 2010 as my initial years because they were some of the earliest I had. I picked 2012 and 2013 to give a year of distance but to maintain a large buffer before COVID affected home values.

1.2.2 Cleaning Loan Data

[6]:
```
loans = loans[loans['purpose'] == 1] # Filter to loans for new purchases
```

[7]:
```
# Define race and ethnicity sets
races_set = {3} # Black
ethnicities_set = {1} # Hispanic

# Extract borrower and coborrower race
borrower_races = loans[[col for col in loans.columns if col.
↪startswith("borrower_race")]]
coborrower_races = loans[[col for col in loans.columns if col.
↪startswith("coborrower_race")]]
```

```
borrower_ethnicities = loans[[col for col in loans.columns if col.
↪startswith("borrower_ethnicity")]]
coborrower_ethnicities = loans[[col for col in loans.columns if col.
↪startswith("coborrower_ethnicity")]]

# Check if any race or ethnicity belongs to minority groups
borrower_minority = borrower_races.isin(races_set).any(axis=1) |␣
↪borrower_ethnicities.isin(ethnicities_set).any(axis=1)
coborrower_minority = coborrower_races.isin(races_set).any(axis=1) |␣
↪coborrower_ethnicities.isin(ethnicities_set).any(axis=1)

# Assign counting columns for quantity of move-ins of each type
loans["early_mmi"] = np.where((loans["year"] < 2011) & (borrower_minority &␣
↪coborrower_minority), 1,
                              np.where((loans["year"] < 2011) &␣
↪(borrower_minority | coborrower_minority), 1, 0))
loans["late_mmi"] = np.where((loans["year"] > 2011) & (borrower_minority &␣
↪coborrower_minority), 1,
                             np.where((loans["year"] > 2011) &␣
↪(borrower_minority | coborrower_minority), 1, 0))

loans["early_move_ins"] = np.where(loans["year"] < 2011, 1, 0)
loans["late_move_ins"] = np.where(loans["year"] > 2011, 1, 0)
```

Finally, we add a column with the full census tract code for future merging

[8]:
```
loans['longtract'] = (
    loans['state_fips'].astype(str).str.zfill(2) +
    loans['county_fips'].astype(str).str.zfill(3) +
    loans['census_tract'].apply(lambda x: f"{int(x):06d}" if x == int(x) else␣
↪f"{x:06.2f}".replace('.', ''))
)
```

1.2.3 Merging

To truly understand the rate of minority move-ins, we must control for neighborhood demographics. The **2020 US Census Data** provides this information (total population, white population, black population)

[9]:
```
cols_tract = ['YEAR', 'ZCTAA','U7J001', 'U7J002', 'U7J003', 'U7J004', 'U7J005',␣
↪'U7J006', 'U7J007', 'U7J008']
tract_data = pd.read_csv('data/census/nhgis0009_ds258_2020_zcta.csv')[cols_tract]
tract_data.rename(columns={
    'YEAR': 'year',
    'ZCTAA': 'zip',
    'U7J001': 'total_pop',
    'U7J002': 'white',
```

```
    'U7J003': 'black',
    'U7J004': 'native_american',
    'U7J005': 'asian',
    'U7J006': 'pacific_islander',
    'U7J007': 'other_race',
    'U7J008': 'two_or_more'}, inplace=True)
```

Next, we load in the Zillow data for information about home prices

```
[10]: zhvi = pd.read_csv('data/zhvi/Zip_zhvi_uc_sfrcondo_tier_0.33_0.67_sm_sa_month␣
       ↪(1).csv')
```

The Zillow data is based on zip-code, but all our other data is based on census tract. As a result, we use a Crosswalk File from HUD.

```
[11]: crosswalk = pd.read_excel('data/census/ZIP_TRACT_122024.xlsx')
```

Next, we merge the loan data into our crosswalk. We aggregate on each census tract.

```
[12]: loans_grouped = loans.groupby(['longtract', 'msa_code', 'state_fips',␣
       ↪'county_fips'])[['early_mmi', 'late_mmi', 'early_move_ins', 'late_move_ins']].
       ↪sum().reset_index()
      # loans_grouped['longtract'] = loans_grouped['longtract'].astype(str).str.
       ↪zfill(11)
      crosswalk['TRACT'] = crosswalk['TRACT'].astype(str).str.zfill(11)
      # loans_grouped['longtract'] = loans_grouped['longtract'].str.rstrip('.0')
      loans_crosswalk = loans_grouped.merge(crosswalk, left_on='longtract',␣
       ↪right_on='TRACT', how='outer')
```

Now that we have loan data, we merge with our **Census data**

```
[13]: loans_census = loans_crosswalk.merge(tract_data, left_on='ZIP', right_on='zip',␣
       ↪how='left')
      threshold_white = 0.0  # We only look at areas that are majority white
      loans_census_filter = loans_census[(loans_census['white'] /␣
       ↪loans_census['total_pop'] > threshold_white)]
```

Now we merge **Zillow data**, and find the change in home values of an area from 2010 to 2019 (End of early move-in data until COVID)

```
[14]: loans_zhvi = loans_census_filter.merge(zhvi, left_on='zip',␣
       ↪right_on='RegionName', how='left')
      loans_zhvi[f'avg_value_2010'] = loans_zhvi[[f'2010-10-31', f'2010-11-30',␣
       ↪f'2010-12-31']].mean(axis=1)
      for year in range(2011, 2020):
          cols_year = [col for col in loans_zhvi.columns if col.startswith(f"{year}-")]
          loans_zhvi[f'avg_value_{year}'] = loans_zhvi[[f'{year}-10-31',␣
       ↪f'{year}-11-30', f'{year}-12-31']].mean(axis=1)
```

```
    loans_zhvi[f'value_ratio_{year}'] = loans_zhvi[f'avg_value_{year}'] /␣
 ↪loans_zhvi[f'avg_value_2010']
loans_zhvi = loans_zhvi.drop(columns=[col for col in loans_zhvi.columns if col[:
 ↪4].isdigit()])
```

[15]:
```
loans_zhvi['fips_code'] = (
    loans_zhvi['state_fips'].fillna('').astype(str).str.split('.').str[0].str.
 ↪zfill(2) +
    loans_zhvi['county_fips'].fillna('').astype(str).str.split('.').str[0].str.
 ↪zfill(3)
)
```

Next, we control for metropolitan area.

[16]:
```
for year in range(2011, 2020):
    loans_zhvi[f'msa_avg_value_ratio_{year}'] = loans_zhvi.
 ↪groupby('msa_code')[f'value_ratio_{year}'].transform('mean')
    loans_zhvi[f'value_diff_{year}'] = loans_zhvi[f'value_ratio_{year}'] -␣
 ↪loans_zhvi[f'msa_avg_value_ratio_{year}']
```

Now we add **income data**

[17]:
```
income_data = pd.DataFrame({
    'med_hh_inc' : pd.read_csv('data/income/nhgis0011_csv/
 ↪nhgis0011_ds262_20225_zcta.csv').set_index('NAME_M')['AQP6E001'],
    'per_cap_inc' : pd.read_csv('data/income/nhgis0011_csv/
 ↪nhgis0011_ds267_20235_zcta.csv').set_index('NAME_M')['ASRTE001'],
    'gini_index' : pd.read_csv('data/income/nhgis0011_csv/
 ↪nhgis0011_ds268_20235_zcta.csv').set_index('NAME_M')['AS9QE001']}
)
income_data.index = income_data.index.str.replace(r'ZCTA5 ', '', regex=True)
income_data.index = income_data.index.astype(np.float64)
income_data.index.name = 'zip'
```

Now, we add **election results**

[18]:
```
election_data = pd.read_csv('data\election-context-2018.csv')
election_data['fips'] = election_data['fips'].astype(str).str.zfill(5)
type(loans_zhvi['fips_code'][2])
election_data['votes'] = election_data['obama12'] + election_data['romney12'] +␣
 ↪election_data['otherpres12']
election_data['pct_rep'] = election_data['romney12'] / election_data['votes'] #␣
 ↪Percent of county zip code is in that voted romney
election_data['pct_dem'] = election_data['obama12'] / election_data['votes']
election_data['pct_vote'] = election_data['votes'] /␣
 ↪election_data['total_population']
```

```
<>:1: SyntaxWarning: invalid escape sequence '\e'
<>:1: SyntaxWarning: invalid escape sequence '\e'
```

```
C:\Users\thest\AppData\Local\Temp\ipykernel_32012\1634695877.py:1:
SyntaxWarning: invalid escape sequence '\e'
  election_data = pd.read_csv('data\election-context-2018.csv')
```

[19]:
```
data_table = loans_zhvi.groupby('zip').agg({
    'total_pop': 'sum',
    'msa_code': 'min',
    'fips_code': 'max',
    'early_mmi': 'sum',
    'late_mmi': 'sum',
    'early_move_ins': 'sum',
    'late_move_ins': 'sum',
    'total_pop' : 'mean',
    'white': 'mean',
    'black': 'mean',
    'other_race': 'sum',
    'two_or_more': 'sum',
    'avg_value_2010': 'mean',
    'avg_value_2011': 'mean',
    'value_ratio_2011': 'mean',
    'avg_value_2012': 'mean',
    'value_ratio_2012': 'mean',
    'avg_value_2013': 'mean',
    'value_ratio_2013': 'mean',
    'avg_value_2014': 'mean',
    'value_ratio_2014': 'mean',
    'avg_value_2015': 'mean',
    'value_ratio_2015': 'mean',
    'avg_value_2016': 'mean',
    'value_ratio_2016': 'mean',
    'avg_value_2017': 'mean',
    'value_ratio_2017': 'mean',
    'avg_value_2018': 'mean',
    'value_ratio_2018': 'mean',
    'avg_value_2019': 'mean',
    'value_ratio_2019': 'mean',
    'msa_avg_value_ratio_2011': 'mean',
    'value_diff_2011': 'mean',
    'msa_avg_value_ratio_2012': 'mean',
    'value_diff_2012': 'mean',
    'msa_avg_value_ratio_2013': 'mean',
    'value_diff_2013': 'mean',
    'msa_avg_value_ratio_2014': 'mean',
    'value_diff_2014': 'mean',
    'msa_avg_value_ratio_2015': 'mean',
    'value_diff_2015': 'mean',
    'msa_avg_value_ratio_2016': 'mean',
```

```
    'value_diff_2016': 'mean',
    'msa_avg_value_ratio_2017': 'mean',
    'value_diff_2017': 'mean',
    'msa_avg_value_ratio_2018': 'mean',
    'value_diff_2018': 'mean',
    'msa_avg_value_ratio_2019': 'mean',
    'value_diff_2019': 'mean',
}).reset_index()
data_table = data_table.join(income_data, on='zip')
data_table = data_table.merge(election_data, left_on='fips_code',␣
 ↪right_on='fips', how='left')
data_table['early_mmi_ratio'] = data_table['early_mmi'] /␣
 ↪data_table['early_move_ins']
data_table['late_mmi_ratio'] = data_table['late_mmi'] /␣
 ↪data_table['late_move_ins']
data_table['treatment_size'] = data_table['late_mmi_ratio'] -␣
 ↪data_table['early_mmi_ratio'] # Positive: MMI higher in late period
data_table['min_share_before'] = data_table['black'] / data_table['total_pop']
data_table['change_in_min_share'] = data_table['late_mmi_ratio'] -␣
 ↪data_table['min_share_before']
```

1.3 Summary Statistics

In our `filtered_data` table, we have information about our explanatory variables (demographics), our independent variables (initial property value), and our outcome variable (final property value)

[20]:
```
from IPython.display import display, HTML

prop_values_summary = data_table[['early_mmi','late_mmi','early_move_ins',␣
 ↪'late_move_ins']].describe().round(2)
prop_values_summary.columns = ['Minority Move-Ins (Initial Period)','Minority␣
 ↪Move-Ins (Treatment Period)', 'Total Move-Ins (Initial Period)', 'Total␣
 ↪Move-Ins (Treatment Period)']
prop_values_summary.index = ['Count', 'Mean', 'Standard Deviation', 'Minimum',␣
 ↪'25th Percentile', 'Median', '75th Percentile', 'Maximum']
prop_values_summary.loc['Range'] = prop_values_summary.loc['Maximum'] -␣
 ↪prop_values_summary.loc['Minimum']
prop_values_summary.loc['Count'] = prop_values_summary.loc['Count'].astype(int)

# Styling
styled_prop_vals = prop_values_summary.style.set_caption("Table 1a: Move-In and␣
 ↪Minority Move-In by Zip Code")

# Apply table styles
styled_prop_vals = styled_prop_vals.set_table_attributes('style="width: 50%;␣
 ↪margin: auto;"')
```

```
# Apply formatting
prop_values_summary['Minority Move-Ins (Initial Period)'] =␣
↪prop_values_summary['Minority Move-Ins (Initial Period)'].astype(int)
prop_values_summary['Minority Move-Ins (Treatment Period)'] =␣
↪prop_values_summary['Minority Move-Ins (Treatment Period)'].astype(int)
prop_values_summary['Total Move-Ins (Initial Period)'] =␣
↪prop_values_summary['Total Move-Ins (Initial Period)'].astype(int)
prop_values_summary['Total Move-Ins (Treatment Period)'] =␣
↪prop_values_summary['Total Move-Ins (Treatment Period)'].astype(int)

# Caption formatting
styled_prop_vals = styled_prop_vals.set_table_styles([{
    'selector': 'caption',
    'props': [
        ('caption-side', 'top'),
        ('font-size', '20px'),
        ('font-style', 'italic'),
        ('text-align', 'center'),
        ('color', '#222')
    ]
}])

latex_code = prop_values_summary.to_latex(
    caption="Table 2: Summary Statistics of Home Value Changes by Year",
    label="tab:summary_stats",
    float_format="%.10f",
    bold_rows=True
)
# print(latex_code)
styled_prop_vals
```

Table 1a: Summary Statistics for Minority Move-ins (all geographies)

	MMI (I)	**MMI (T)**	**MI (I)**	**MI (T)**
Count	32208	32208	32208	32208
Mean	5	8	69	117
Standard Deviation	16	20	121	192
Minimum	0	0	0	0
25th Percentile	0	0	1	11
Median	0	1	21	38
75th Percentile	3	6	79	132
Maximum	403	615	1460	2338
Range	403	615	1460	2338

Where MMI = Minority Move-Ins, MI = Move-Ins, (I) is the initial period, and (T) is the treatment period

This table shows one of the difficulties of the project, many zip codes either do not have many or have no minority-move ins (or any move-ins). As you can see, in the initial period, the 25th percentile zip code had zero minority move-ins. For all following tables and graphs, we are only looking at zip codes that had move-ins during both periods. This could lead to bias, but it aligns with the research question, so it should not meaningfully change our conclusion.

[21]:

```python
prop_values_summary = data_table.query("msa_code != 99999")[['early_mmi','late_mmi','early_move_ins', 'late_move_ins']].describe().round(2)
prop_values_summary.columns = ['Minority Move-Ins (Initial Period)','Minority Move-Ins (Treatment Period)', 'Total Move-Ins (Initial Period)', 'Total Move-Ins (Treatment Period)']
prop_values_summary.index = ['Count', 'Mean', 'Standard Deviation', 'Minimum', '25th Percentile', 'Median', '75th Percentile', 'Maximum']
prop_values_summary.loc['Range'] = prop_values_summary.loc['Maximum'] - prop_values_summary.loc['Minimum']
prop_values_summary.loc['Count'] = prop_values_summary.loc['Count'].astype(int)

# Styling
styled_prop_vals = prop_values_summary.style.set_caption("Table 1b: Move-In and Minority Move-In by Zip Code (only MSA)")

# Apply table styles
styled_prop_vals = styled_prop_vals.set_table_attributes('style="width: 50%; margin: auto;"')

# Apply formatting
prop_values_summary['Minority Move-Ins (Initial Period)'] = prop_values_summary['Minority Move-Ins (Initial Period)'].astype(int)
prop_values_summary['Minority Move-Ins (Treatment Period)'] = prop_values_summary['Minority Move-Ins (Treatment Period)'].astype(int)
prop_values_summary['Total Move-Ins (Initial Period)'] = prop_values_summary['Total Move-Ins (Initial Period)'].astype(int)
prop_values_summary['Total Move-Ins (Treatment Period)'] = prop_values_summary['Total Move-Ins (Treatment Period)'].astype(int)

# Caption formatting
styled_prop_vals = styled_prop_vals.set_table_styles([{
    'selector': 'caption',
    'props': [
        ('caption-side', 'top'),
        ('font-size', '20px'),
        ('font-style', 'italic'),
        ('text-align', 'center'),
        ('color', '#222')
    ]
}])
```

```
latex_code = prop_values_summary.to_latex(
    caption="Table 1b: Summary Statistics of Home Value Changes by Year",
    label="tab:summary_stats",
    float_format="%.10f",
    bold_rows=True
)
# print(latex_code)
styled_prop_vals
```

Table 1b: Summary Statistics of Home Value Changes by Year (only in-MSA ZIPs)

	MMI (I)	MMI (T)	MI (I)	MI (T)
Count	19847	19847	19847	19847
Mean	8	12	100	167
Standard Deviation	20	25	142	227
Minimum	0	0	0	0
25th Percentile	0	0	6	19
Median	1	3	44	76
75th Percentile	8	14	137	230
Maximum	403	615	1460	2338
Range	403	615	1460	2338

Where MMI = Minority Move-Ins, MI = Move-Ins, (I) is the initial period, and (T) is the treatment period

This table shows that, while necessary for robustness (controlling for MSA-level appreciation), restricting our sample to MSAs also increases the quantity of minority move-ins and move-ins.

[22]:
```
# Select value difference columns
value_diff_cols = [col for col in data_table.columns if col.
 ↪startswith('value_diff_')]

# Create readable column names
renamed_cols = {col: f"Home Value Change {col.split('_')[-1]}" for col in␣
 ↪value_diff_cols}

# Compute summary statistics
summary_stats = data_table[value_diff_cols].describe().round(10)

# Convert to DataFrame (ensuring correct format)
summary_df = pd.DataFrame(summary_stats)

# Rename columns for better readability
summary_df = summary_df.rename(columns=renamed_cols)

# Rename index for better readability
summary_df.index.names = ['Statistic']
# Adjust rounding for better readability
```

Table 2: Summary Statistics of Average Home Value Change (%) by Year

Statistic	count	mean	std	min	25%	50%	75%	max
2011	18246	0.003	0.039	-0.392	-0.015	0.005	0.023	0.362
2012	18248	0.003	0.062	-0.503	-0.028	0.004	0.037	0.489
2013	18248	0.001	0.083	-0.712	-0.042	0.000	0.045	0.885
2014	18248	-0.000	0.101	-0.801	-0.054	-0.002	0.049	0.973
2015	18248	-0.002	0.125	-0.907	-0.067	-0.006	0.055	1.117
2016	18248	-0.005	0.148	-1.054	-0.083	-0.012	0.060	1.669
2017	18248	-0.007	0.175	-1.147	-0.102	-0.017	0.066	1.968
2018	18248	-0.011	0.210	-1.240	-0.121	-0.024	0.072	2.496
2019	18248	-0.012	0.234	-1.251	-0.137	-0.029	0.081	3.036

```
# Ensure 'Count' is integer
summary_df.loc['count'] = summary_df.loc['count'].astype(int)
summary_df = summary_df.T
# Styling

# Ensure 'count' is integer
summary_df['count'] = summary_df['count'].astype(int)

# Display the table
latex_code = summary_df.to_latex(
    caption="Table 2: Summary Statistics of Home Value Changes by Year",
    label="tab:summary_stats",
    float_format="%.10f",
    bold_rows=True
)
# print(latex_code)
summary_df
```

It should not be surprising that the median and mean are basically zero. Recall that we are normalizing home-value change to the metropolitan area's home-value change. It should also be unsurprising that the ranges increase over time. The difference between a neighborhood and metro area will be more obvious over time. This table has significantly fewer observations than the table above, due to the unpredictable availability of the Zillow data. The move-in data is not significantly different between the zip codes with Zillow data and the zip codes without, so we will drop the zip codes without Zillow data.

[23]:
```
data_table = data_table[data_table['per_cap_inc'] >= 0]
data_table = data_table[data_table['gini_index'] >= 0]

income_summary = data_table[['med_hh_inc', 'per_cap_inc', 'gini_index']].
 ↪describe()

income_summary.columns = ['Median Household Income', 'Per Capita Income', 'Gini␣
 ↪Index']
```

```
income_summary.index = ['Count', 'Mean', 'Standard Deviation', 'Minimum', '25th
↪Percentile', 'Median', '75th Percentile', 'Maximum']

# Format numbers to be more readable
income_summary['Median Household Income'] = income_summary['Median Household
↪Income'].astype(int)
income_summary['Per Capita Income'] = income_summary['Per Capita Income'].
↪astype(int)
income_summary['Gini Index'] = income_summary['Gini Index'].round(4)

# Display the table with styling
styled_table = income_summary.style.set_caption("Table 3: Income Statistics by
↪Zip Code")

# Apply table styles
styled_table = styled_table.set_table_attributes('style="width: 50%; margin:
↪auto;"')

# Display the table
print(styled_table.to_latex())
```

Table 3: Summary Statistics for Income

	Median Household Income	Per Capita Income	Gini Index
Count	30562	32208	32208
Mean	73170	39472	0.414700
Standard Deviation	31347	18844	0.081300
Minimum	2499	421	0.001000
25th Percentile	53500	28762	0.376500
Median	67028	35573	0.418200
75th Percentile	85316	44910	0.460100
Maximum	250001	419459	1.000000

Income is widely ranging, as is Gini Index. Most outliers are in small zip codes that are not relevant to our analysis. It is not surprising that the average values for median houeshold income and per capita income are slightly lower than nationwide average. Both variables are skewed heavily right, affecting the nationwide statistics.

[24]:
```
# Update the variable names based on your data
data_table['pct_white'] = data_table['white'] / data_table['total_pop']
demographic_summary = data_table[['pct_white', 'pct_rep']].describe()

# Create summary statistics for demographic-related columns
demographic_summary.columns = ['White Population', 'Pct Republican']
demographic_summary.index = ['Count', 'Mean', 'Standard Deviation', 'Minimum',
↪'25th Percentile', 'Median', '75th Percentile', 'Maximum']
```

```python
demographic_summary.loc['Range'] = demographic_summary.loc['Maximum'] -␣
↪demographic_summary.loc['Minimum']
demographic_summary.loc['Count'] = demographic_summary.loc['Count'].astype(int)

# Styling
styled_demographics = demographic_summary.style.set_caption("Table 4:␣
↪Demographic Statistics by Zip Code")

# Apply table styles
styled_demographics = styled_demographics.set_table_attributes('style="width:␣
↪70%; margin: auto;"')

# Caption formatting
styled_demographics = styled_demographics.set_table_styles([{
    'selector': 'caption',
    'props': [
        ('caption-side', 'top'),
        ('font-size', '20px'),
        ('font-style', 'italic'),
        ('text-align', 'center'),
        ('color', '#222')
    ]
}])

print(styled_demographics.to_latex())
```

Table 4: Summary Statistics for Demographics and Political Affiliation

	% White	% Republican
Count	32208	29540
Mean	0.765480	0.539260
Standard Deviation	0.226515	0.152746
Minimum	0.000762	0.071934
25th Percentile	0.669562	0.432962
Median	0.859255	0.544754
75th Percentile	0.931172	0.650237
Maximum	1.000000	0.932903
Range	0.999238	0.860969

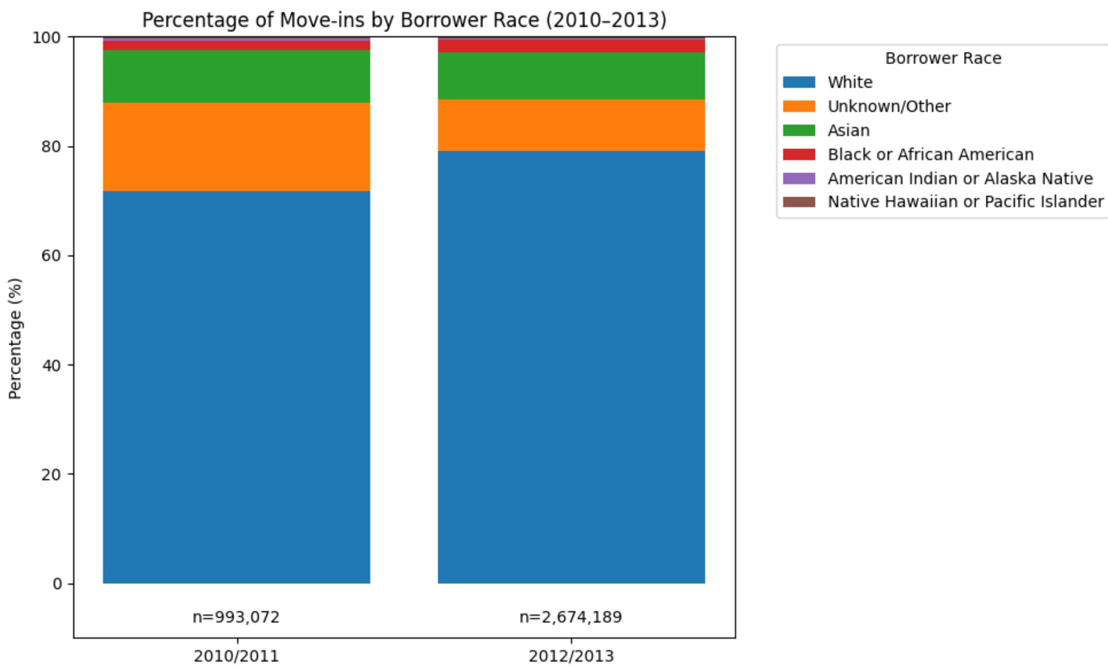
The median zip code is quite white (this is because zip codes are distributed for geographic ease of the USPS, so there are many rural zip codes).

1.4 Plots & Figures

1.4.1 Loan race data

[25]:
```python
race_labels = {
    1: "American Indian or Alaska Native",
    2: "Asian",
    3: "Black or African American",
    4: "Native Hawaiian or Pacific Islander",
    5: "White",
    6: "Unknown/Other",
    7: "Unknown/Other",
    9: "Unknown/Other"
}

relevant_years = [2010, 2011, 2012, 2013]
filtered_loans = loans[loans['year'].isin(relevant_years)].copy()

filtered_loans['borrower_race_clean'] = filtered_loans['borrower_race1'].
 ↪replace(race_labels)
year_groups = {
    '2010/2011': ([2010, 2011], 'early_move_ins'),
    '2012/2013': ([2012, 2013], 'late_move_ins')
}

race_distribution = {}
total_counts = {}

for label, (years, movein_col) in year_groups.items():
    group = filtered_loans[filtered_loans['year'].isin(years)]

    race_sums = group.groupby('borrower_race_clean')[movein_col].sum()
    total = race_sums.sum()

    race_distribution[label] = (race_sums / total * 100).fillna(0)
    total_counts[label] = int(total)

distribution_df = pd.DataFrame(race_distribution).fillna(0)
distribution_df = distribution_df.loc[distribution_df.sum(axis=1).
 ↪sort_values(ascending=False).index]

plt.figure(figsize=(10, 6))
bottom = [0] * len(year_groups)
colors = plt.cm.tab10.colors  # color palette
labels = list(year_groups.keys())

for i, (race, row) in enumerate(distribution_df.iterrows()):
```

```
    plt.bar(labels, row, bottom=bottom, label=race, color=colors[i %
 ↪len(colors)])
    bottom = [sum(x) for x in zip(bottom, row)]

for idx, label in enumerate(labels):
    plt.text(idx, -5, f'n={total_counts[label]:,}', ha='center', va='top',
 ↪fontsize=10)

plt.ylabel('Percentage (%)')
plt.title('Percentage of Move-ins by Borrower Race (2010-2013)')
plt.ylim(-10, 100)
plt.legend(title='Borrower Race', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.show()
```

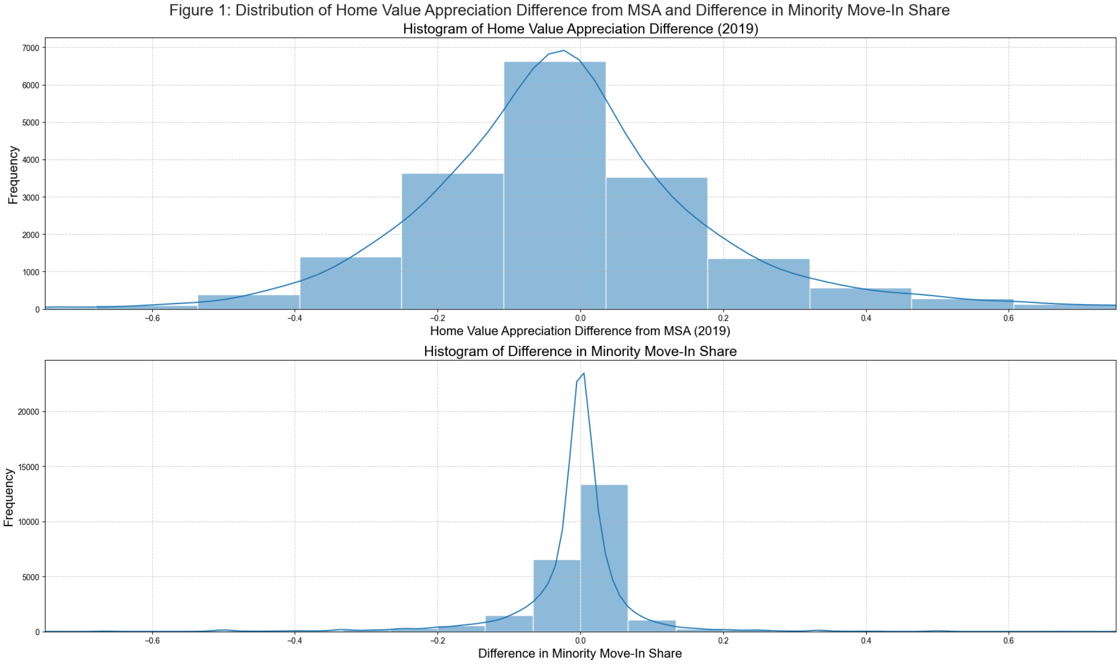


This chart shows the change in primary borrower primary racial demographics from the initial period to the treatment period over the whole sample of loans. You can see that only a slim proportion are classified as Black or African American, with the vast majority being White or Asian. This is consistent with national homeownership demographics. For the purposes of this analysis, a move in is classified as "minority" if either the borrower or coborrower are either partially African American or are hispanic.

1.4.2 Histogram of X and Y

[26]:
```
fig, axs = plt.subplots(2, 1, figsize=(20, 12))
sns.set_style("whitegrid")

sns.histplot(data_table['value_diff_2019'], bins=30, kde=True, ax=axs[0])
axs[0].set_xlabel('Home Value Appreciation Difference from MSA (2019)',␣
 ↪fontsize=16)
axs[0].set_ylabel('Frequency', fontsize=16)
axs[0].set_title('Histogram of Home Value Appreciation Difference (2019)',␣
 ↪fontsize=18)
axs[0].set_xlim(-.75, .75)
axs[0].grid(True, linestyle="--", alpha=0.6)

sns.histplot(data_table['treatment_size'], bins=30, kde=True, ax=axs[1])
axs[1].set_xlabel('Difference in Minority Move-In Share', fontsize=16)
axs[1].set_ylabel('Frequency', fontsize=16)
axs[1].set_xlim(-.75, .75)
axs[1].set_title('Histogram of Difference in Minority Move-In Share',␣
 ↪fontsize=18)
axs[1].grid(True, linestyle="--", alpha=0.6)

fig.suptitle('Figure 1: Distribution of Home Value Appreciation Difference from␣
 ↪MSA and Difference in Minority Move-In Share', fontsize=20)

plt.tight_layout()
plt.subplots_adjust(top=0.93)
plt.show()
```

Figure 1: Distribution of Home Value Appreciation Difference from MSA and Difference in Minority Move-In Share



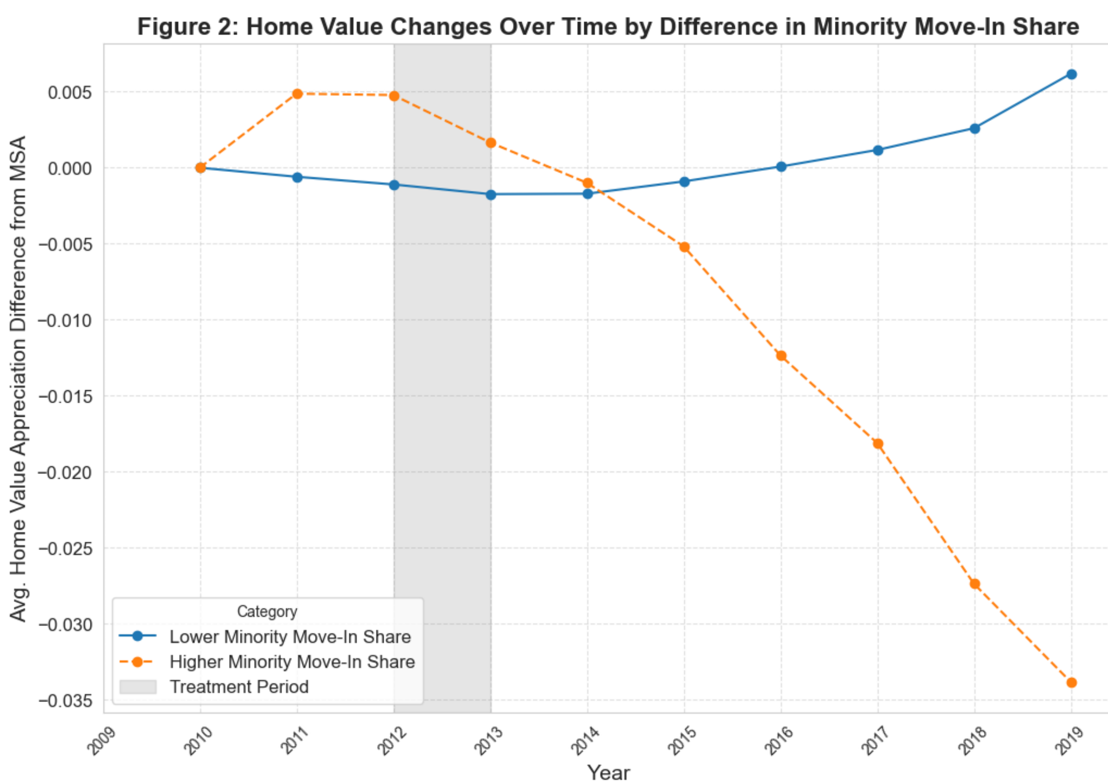
As shown in the summary tables above, the data has an incredibly large range. This plot shows how incredibly wide the range is in home value appreciation and minority move-in share (and how incredibly concentrated the difference in minority move-in share is around zero). This figure shows how large the distribution is. In the next few plots we will attempt to answer my research question and see to what extent the distribution on the left can be explained by the distribution on the right.

1.4.3 Time Series of Y With Discrete X

[27]:
```
plt.figure(figsize=(12, 8))

data_table['early_late_category'] = pd.cut(
    data_table['treatment_size'],
    bins=[-float('inf'), 0.0, float('inf')],
    labels=['lower', 'higher']
)

data_table['value_diff_2010'] = 0
melted_data = data_table.melt(
    id_vars=['early_late_category'],
    value_vars=[col for col in data_table.columns if col.
↪startswith('value_diff_')],
    var_name='year', value_name='value_diff'
)
melted_data['year'] = melted_data['year'].str.extract('(\d+)').astype(int)

agg_data = melted_data.groupby(['early_late_category', 'year'],␣
↪observed=False)['value_diff'].agg(['mean', 'std']).reset_index()

colors = {"lower": "#1f77b4", "higher": "#ff7f0e"}  # Blue for low, Orange for␣
↪high
linestyles = {"lower": "solid", "higher": "dashed"}  # Solid & dashed lines

# Plot data with shaded error area
for category, data in agg_data.groupby('early_late_category', observed=False):
    plt.plot(
        data['year'], data['mean'],
        label=f'{category.capitalize()} Minority Move-In Share',
        color=colors[category], linestyle=linestyles[category], marker='o'
    )
#     # Add shaded error area
#     plt.fill_between(
#         data['year'],
#         data['mean'] - data['std'],
#         data['mean'] + data['std'],
#         color=colors[category], alpha=0.2
```

```
#     )

# Highlight treatment period
plt.axvspan(2012, 2013, color='gray', alpha=0.2, label='Treatment Period')

# Labels and title
plt.xlabel('Year', fontsize=14)
plt.ylabel('Avg. Home Value Appreciation Difference from MSA', fontsize=14)
plt.title('Figure 2: Home Value Changes Over Time by Difference in Minority␣
 ↪Move-In Share', fontsize=16, fontweight='bold')

plt.legend(title='Category', fontsize=12)
plt.xticks(range(agg_data['year'].min()-1, agg_data['year'].max() + 1),␣
 ↪rotation=45)
plt.yticks(fontsize=12)

plt.grid(True, linestyle="--", alpha=0.6)
plt.show()
```

```
<>:15: SyntaxWarning: invalid escape sequence '\d'
<>:15: SyntaxWarning: invalid escape sequence '\d'
C:\Users\thest\AppData\Local\Temp\ipykernel_32012\923715123.py:15:
SyntaxWarning: invalid escape sequence '\d'
  melted_data['year'] = melted_data['year'].str.extract('(\d+)').astype(int)
```



This figure shows the difference in outcome between the treatment group (higher minority move-in share) and control group (lower / same minority move-in share). It seems to indicate that the treatment experienced some home-value appreciation relative to their MSA prior to the treatment period. During the treatment period, the treatment group begins to experience a decrase in home values, which continues until the end of the graph. In contrast, the control group has every slightly increasing values for roughly the entire period. This seems to indicate that neighborhoods with a higher minority move-in share have depressed home value appreciation relative to groups without.

1.4.4 Scatterplot of Main X and Y

[29]:
```
# Filter data
filtered_data = data_table[data_table['total_pop'].abs() > 100]

# Define color palette
colors = sns.color_palette("husl", 4)  # Distinct colors for better visibility

# Create subplots
fig, axs = plt.subplots(2, 2, figsize=(15, 10), sharex=True, sharey=True)

years = [2016, 2017, 2018, 2019]
for i, year in enumerate(years):
    ax = axs[i // 2, i % 2]

    # Scatter plot with improved aesthetics
    ax.scatter(
        filtered_data['treatment_size'],
        filtered_data[f'value_diff_{year}'],
        alpha=0.6, color=colors[i], edgecolor='black'
    )
    ax.set_xscale('symlog')

    # Titles and labels
    ax.set_title(f'Difference in Minority Move-In Share vs Change in Home Value␣
↪({year})', fontsize=14, fontweight='bold')
    ax.set_xlabel('Difference in Minority Move-In Share (log scale)',␣
↪fontsize=12)
    ax.set_ylabel(f'Home Value Appreciation Relative to MSA ({year})',␣
↪fontsize=12)
    ax.xaxis.set_major_formatter(plt.FuncFormatter(lambda x, _: f'{x:.2f}'))
    # Grid styling
    ax.grid(True, linestyle="--", alpha=0.5)
# Improve layout
fig.suptitle('Figure 3: Home Value Appreciation and Change in Minority Move-In␣
↪Share by Year', fontsize=16, fontweight='bold')
plt.tight_layout()
```

```
plt.show()
```

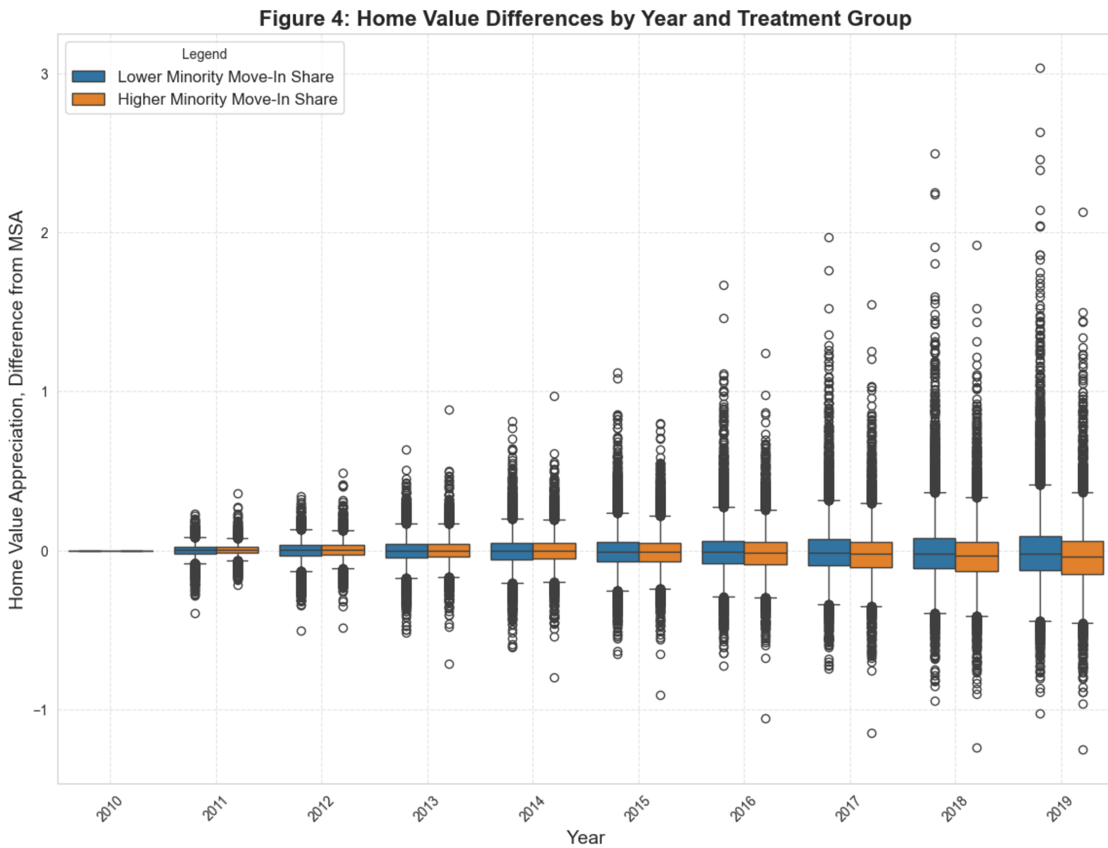
This figure shows something very important not shown in Figure 2, which is how wide the variation is. I chose a log scale to show more detail (as you can see, there are a few far outliers, with most of the data clustered near zero). From these scatterplots, it is clear there is a high degree of variance, but it appears that areas with high treatment are less likely to recieve a large increase in home-value appreciation relative to their MSA than areas with lower treatment. This is more noticable with time.

1.4.5 Box Plot With Discrete X

[30]:
```
data_table['early_late_category'] = pd.cut(data_table['treatment_size'],␣
↪bins=[-float('inf'), 0.00, float('inf')], labels=['Lower Minority Move-In␣
↪Share', 'Higher Minority Move-In Share'])

# Melt the dataframe for easier plotting
melted_data = data_table.melt(id_vars=['early_late_category'], value_vars=[col␣
↪for col in data_table.columns if col.startswith('value_diff_')],␣
↪var_name='year', value_name='value_diff')
# Extract numeric year
melted_data['year'] = melted_data['year'].str.extract('(\d+)').astype(int)
# Compute average value difference per category and year
avg_value_change = melted_data.groupby(['early_late_category', 'year'],␣
↪observed=False)['value_diff'].mean().reset_index()
```

```python
sns.set_style("whitegrid")

# Create figure
plt.figure(figsize=(14, 10))
sns.boxplot(x='year', y='value_diff', hue='early_late_category', data=melted_data, palette={"Lower Minority Move-In Share": "#1f77b4", "Higher Minority Move-In Share": "#ff7f0e"})

# Add labels and title
plt.xlabel('Year', fontsize=14)
plt.ylabel('Home Value Appreciation, Difference from MSA', fontsize=14)
plt.title('Figure 4: Home Value Differences by Year and Treatment Group', fontsize=16, fontweight='bold')
plt.legend(title='Legend', fontsize=12)

# Rotate x-axis labels for better readability
plt.xticks(rotation=45)
plt.grid(True, linestyle="--", alpha=0.5)

# Show plot
plt.show()
```

```
<>:6: SyntaxWarning: invalid escape sequence '\d'
<>:6: SyntaxWarning: invalid escape sequence '\d'
C:\Users\thest\AppData\Local\Temp\ipykernel_32012\4157695199.py:6:
SyntaxWarning: invalid escape sequence '\d'
  melted_data['year'] = melted_data['year'].str.extract('(\d+)').astype(int)
```



This figure shows the extent of the increase in variation across the time period, and that there is a decrease in the median for the higher minority move-in share group. There remains a high degree of variation in home values relative to the metropolitan area, which cannot be explained by the change in minority move-in share, but it does appear that the areas with an increased minority move-in share generally experienced decreased home-value appreciation.

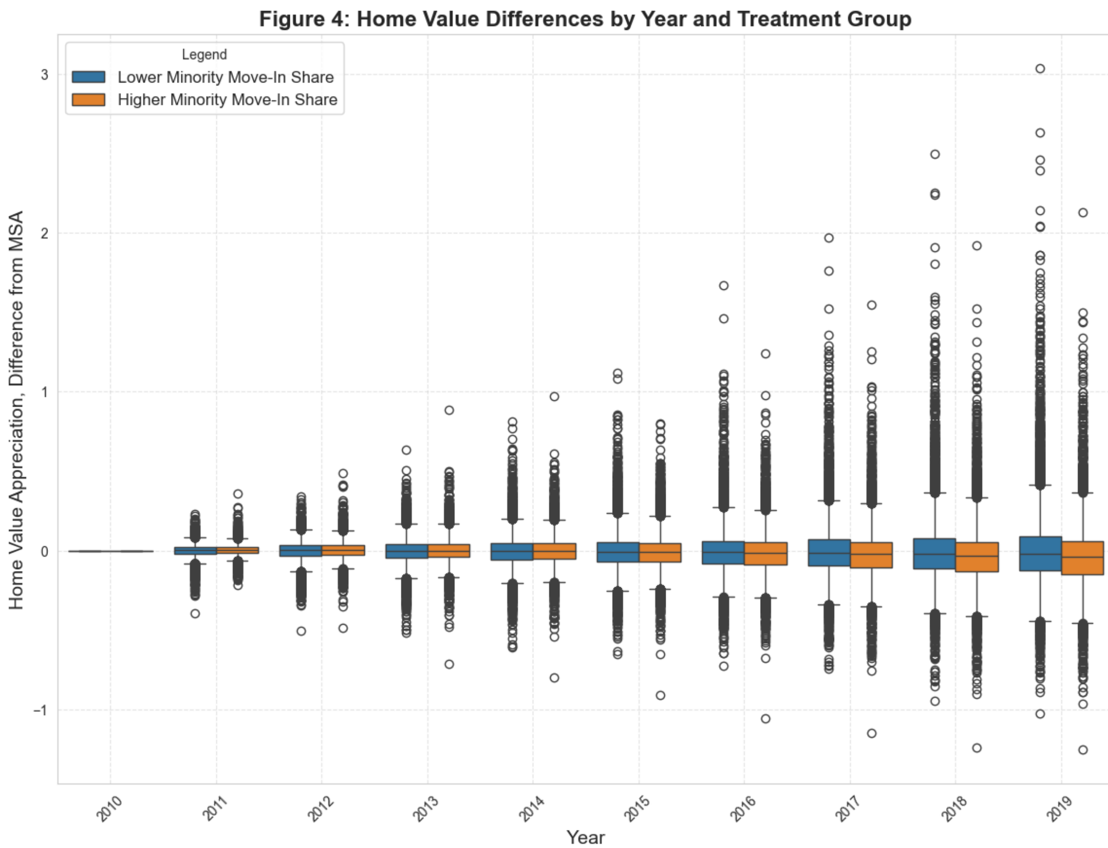
2 Project Two

2.1 The Message

My message is that **while there is significant variation, an increase in the share of move-ins that are from minorities is associated with suppressed home-value appreciation**. It is summed with the final plot above, which shows that there is significant variance but the higher treatment ZIP codes experience fewer high outliers and slightly more significant low outliers. In addition, the higher treatment ZIP codes show a lower median home-value appreciation.

[68]:
```
sns.set_style("whitegrid")

plt.figure(figsize=(14, 10))
```

```
sns.boxplot(x='year', y='value_diff', hue='early_late_category',␣
↪data=melted_data, palette={"Lower Minority Move-In Share": "#1f77b4", "Higher␣
↪Minority Move-In Share": "#ff7f0e"})

plt.xlabel('Year', fontsize=14)
plt.ylabel('Home Value Appreciation, Difference from MSA', fontsize=14)
plt.title('Figure 4: Home Value Differences by Year and Treatment Group',␣
↪fontsize=16, fontweight='bold')
plt.legend(title='Legend', fontsize=12)

plt.xticks(rotation=45)
plt.grid(True, linestyle="--", alpha=0.5)
plt.show()
```



2.2 Maps

[32]:
```
import geopandas as gpd
from matplotlib.patches import Patch
from matplotlib.patches import Rectangle
```

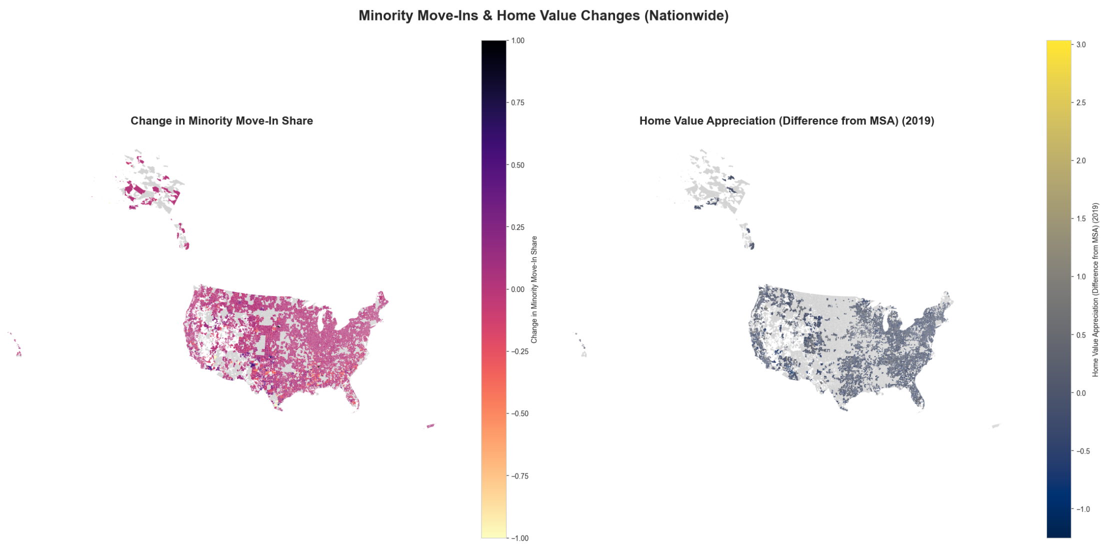
[33]:

```
zips = gpd.read_file('shapefiles/zips/US_zcta_2010.shp')
data_table['zip'] = data_table['zip'].astype(int)
zips['GEOID10'] = zips['GEOID10'].astype(int)
zips = zips.merge(data_table, left_on='GEOID10', right_on='zip')
```

[34]:

```
fig, (gax1, gax2) = plt.subplots(1, 2, figsize=(22, 10), constrained_layout=True)

cmap1 = "magma_r"
plot1 = zips.plot(
    ax=gax1, edgecolor='black', linewidth=0, cmap=cmap1, legend=True,
    column='treatment_size',
    legend_kwds={'label': 'Change in Minority Move-In Share'},
    missing_kwds = {'color':'lightgrey'}
)
gax1.set_title('Change in Minority Move-In Share', fontsize=16,␣
 ↪fontweight='bold', pad=15)
gax1.axis('off')
gax1.set_aspect('equal')

cmap2 = "cividis"
plot2 = zips.plot(
    ax=gax2, edgecolor='black', linewidth=0, cmap=cmap2, legend=True,
    column='value_diff_2019',
    legend_kwds={'label': 'Home Value Appreciation (Difference from MSA)␣
 ↪(2019)'},
    missing_kwds = {'color':'lightgrey'}
)
gax2.set_title('Home Value Appreciation (Difference from MSA) (2019)',␣
 ↪fontsize=16, fontweight='bold', pad=15)
gax2.axis('off')
gax2.set_aspect('equal')

fig.suptitle("Minority Move-Ins & Home Value Changes (Nationwide)",
             fontsize=20, fontweight='bold', y=1.05)

fig.subplots_adjust(wspace=0.3)
plt.show()
```

```
C:\Users\thest\AppData\Local\Temp\ipykernel_32012\2386887725.py:28: UserWarning:
This figure was using a layout engine that is incompatible with subplots_adjust
and/or tight_layout; not calling subplots_adjust.
  fig.subplots_adjust(wspace=0.3)
```

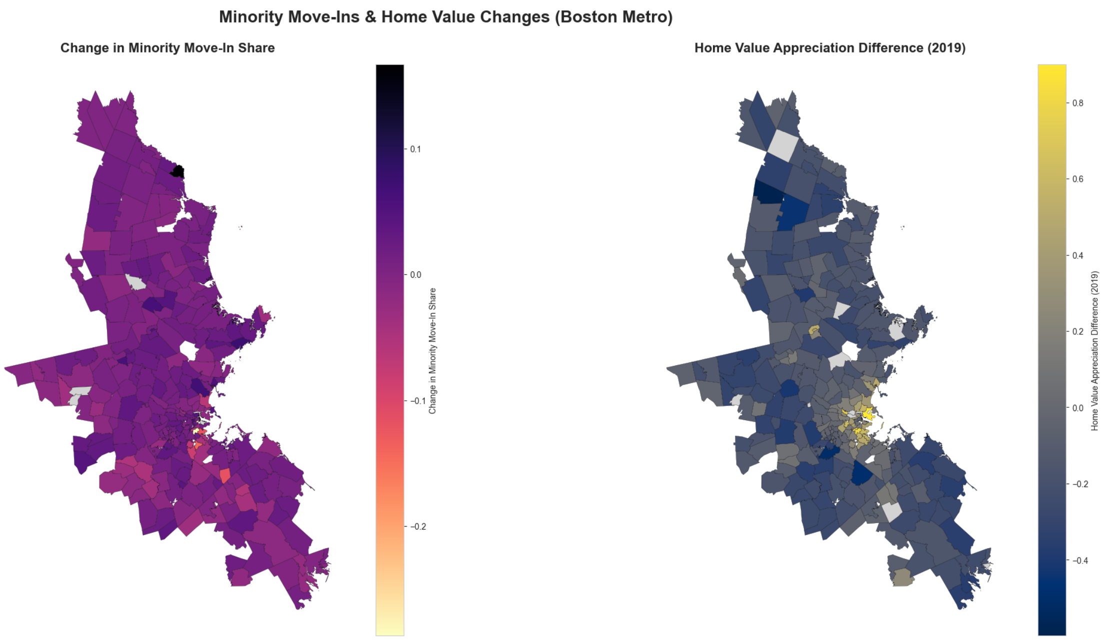


As you can see, this map alone is not very useful, as we are mostly looking at fine effects on a neighborhood level, meaning nationwide data is too broad. In the map below, we zoom in to only look at the Boston MSA. This map also shows that there are many gaps in the home value appreciation difference from MSA. This is because large chunks of the United States are not in MSAs.

[35]:
```
fig, (gax1, gax2) = plt.subplots(1, 2, figsize=(22, 10), constrained_layout=True)

cmap1 = "magma_r"
plot1 = zips.query("msa_code == 14460.0").plot(
    ax=gax1, edgecolor='black', linewidth=0.2, cmap=cmap1, legend=True,
    column='treatment_size',
    legend_kwds={'label': 'Change in Minority Move-In Share'},
    missing_kwds = {'color':'lightgrey'}
)
gax1.set_title('Change in Minority Move-In Share', fontsize=16, fontweight='bold', pad=15)
gax1.axis('off')
gax1.set_aspect('equal')  # Force equal aspect ratio

cmap2 = "cividis"
plot2 = zips.query("msa_code == 14460.0").plot(
    ax=gax2, edgecolor='black', linewidth=0.2, cmap=cmap2, legend=True,
    column='value_diff_2019',
    legend_kwds={'label': 'Home Value Appreciation Difference (2019)'},
    missing_kwds = {'color':'lightgrey'}
)
```

```
gax2.set_title('Home Value Appreciation Difference (2019)', fontsize=16,␣
↪fontweight='bold', pad=15)
gax2.axis('off')
gax2.set_aspect('equal')

fig.suptitle("Minority Move-Ins & Home Value Changes (Boston Metro)",
             fontsize=20, fontweight='bold', y=1.05)

fig.subplots_adjust(wspace=0.3)
plt.show()
```

```
C:\Users\thest\AppData\Local\Temp\ipykernel_32012\3539949142.py:29: UserWarning:
This figure was using a layout engine that is incompatible with subplots_adjust
and/or tight_layout; not calling subplots_adjust.
  fig.subplots_adjust(wspace=0.3)
```

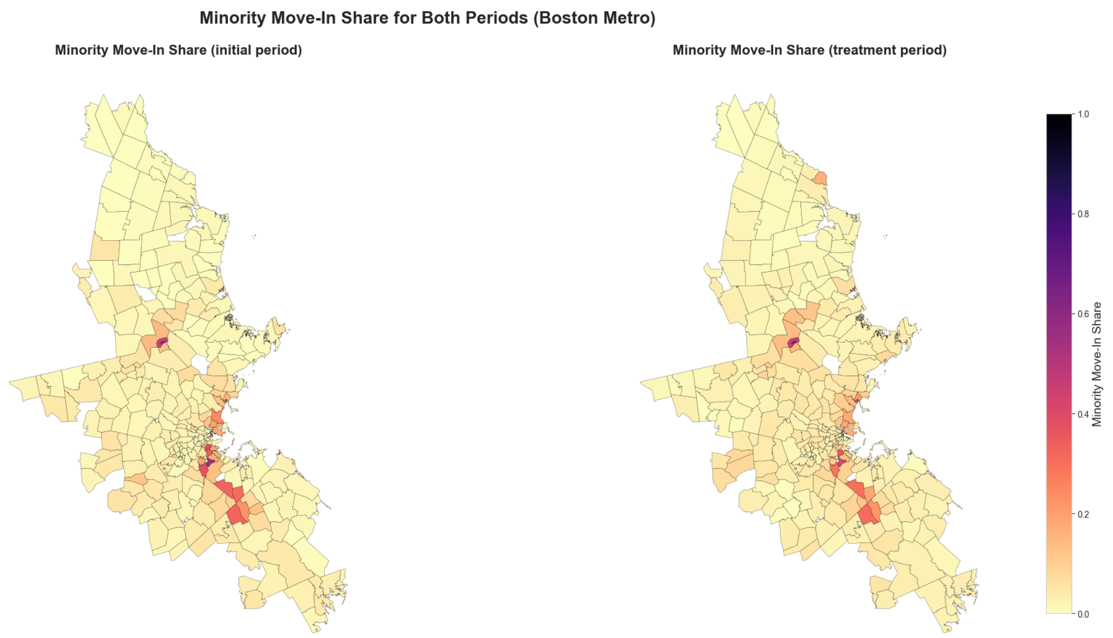


These maps are slightly difficult to gain a conclusion from, as there is large variation in minority move-in share and home value appreciation difference. It appears that several of the areas with suppressed home-value apppreciation difference did have a larger treatment effect applied, but it varies significantly.

[36]:
```
vmin = min(zips['early_mmi_ratio'].min(), zips['late_mmi_ratio'].min())
vmax = max(zips['early_mmi_ratio'].max(), zips['late_mmi_ratio'].max())

fig, (gax1, gax2) = plt.subplots(1, 2, figsize=(22, 10), constrained_layout=True)

# First plot
```

```python
cmap1 = "magma_r"
plot1 = zips.query("msa_code == 14460.0").plot(
    ax=gax1, edgecolor='black', linewidth=0.2, cmap=cmap1,
    column='early_mmi_ratio', legend=False,  # Set legend=False here
    vmin=vmin, vmax=vmax  # Set the color scale to match the common range
)
gax1.set_title('Minority Move-In Share (initial period)', fontsize=16,␣
 ↪fontweight='bold', pad=15)
gax1.axis('off')
gax1.set_aspect('equal')  # Force equal aspect ratio

# Second plot
plot2 = zips.query("msa_code == 14460.0").plot(
    ax=gax2, edgecolor='black', linewidth=0.2, cmap=cmap1,
    column='late_mmi_ratio', legend=False,
    vmin=vmin, vmax=vmax  # Set the color scale to match the common range
)
gax2.set_title('Minority Move-In Share (treatment period)', fontsize=16,␣
 ↪fontweight='bold', pad=15)
gax2.axis('off')
gax2.set_aspect('equal')  # Force equal aspect ratio

# Add a single shared colorbar
sm = plt.cm.ScalarMappable(cmap='magma_r', norm=plt.Normalize(vmin=vmin,␣
 ↪vmax=vmax))
sm.set_array([])  # Only needed for the colorbar to show properly
cbar = fig.colorbar(sm, ax=[gax1, gax2], orientation='vertical', fraction=0.02,␣
 ↪pad=0.04)
cbar.set_label('Minority Move-In Share', fontsize=14)

# Add the overall title
fig.suptitle("Minority Move-In Share for Both Periods (Boston Metro)",␣
 ↪fontsize=20, fontweight='bold', y=1.05)

# Adjust plot spacing
fig.subplots_adjust(wspace=0.3)

plt.show()
```

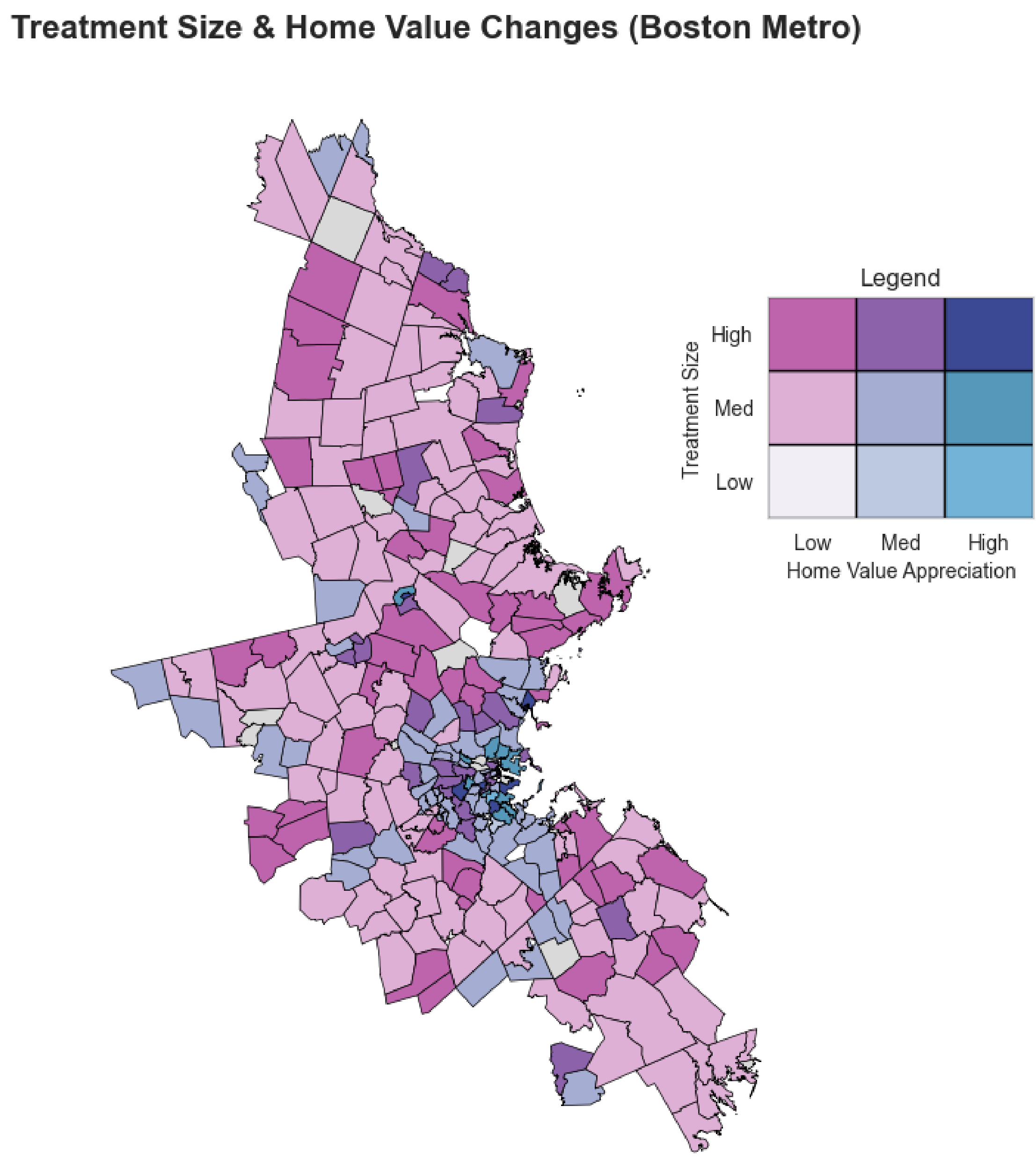
```
C:\Users\thest\AppData\Local\Temp\ipykernel_32012\76696055.py:37: UserWarning:
This figure was using a layout engine that is incompatible with subplots_adjust
and/or tight_layout; not calling subplots_adjust.
  fig.subplots_adjust(wspace=0.3)
```



This map shows the incidence of treatment. It is unsurprising that areas with a high minority move-in share in the initial period also experience a high minority move-in share in the treatment period. We are interested in the difference in share, to assess a 'treatment value' to see how it is related to future change in home values. This map shows that there is reaonably significant variation in minority move-in share between the periods and ZCTAs. This is critical for any future analysis.

[37]:
```
def normalize(series):
    return (series - series.min()) / (series.max() - series.min())

msa_code = 14460.0
df = zips.query("msa_code == @msa_code").copy()

# Normalize values
df['norm_movein'] = normalize(df['treatment_size'])
df['norm_value'] = normalize(df['value_diff_2019'])

# Define a distinct bivariate colormap (3x3 grid)
bivariate_colormap = [
    ['#f1eef6', '#bdc9e1', '#73b3d8'],  # Low move-in
    ['#dfb0d6', '#a5add3', '#5698b9'],  # Medium move-in
    ['#be64ac', '#8c62aa', '#3b4994'],  # High move-in
]

def bivariate_color(movein, value):
    if np.isnan(movein) or np.isnan(value):
        return '#d9d9d9'  # fallback for NaNs
```

```
    row = min(2, int(movein * 2.999))
    col = min(2, int(value * 2.999))
    return bivariate_colormap[row][col]

# Apply color
df['color'] = df.apply(lambda row: bivariate_color(row['norm_movein'],␣
 ↪row['norm_value']), axis=1)

# Plot
fig, ax = plt.subplots(figsize=(12, 10))
df.plot(ax=ax, color=df['color'], edgecolor='black', linewidth=0.5,
        missing_kwds={'color': 'lightgray'})

# Title
ax.set_title("Treatment Size & Home Value Changes (Boston Metro)",
             fontsize=16, fontweight='bold', pad=15)
ax.axis('off')

# Legend
fig.subplots_adjust(right=1.0)
legend_size = 0.15
legend_ax = fig.add_axes([0.75, 0.65 - legend_size / 2, legend_size,␣
 ↪legend_size])

for i in range(3):
    for j in range(3):
        color = bivariate_colormap[i][j]
        legend_ax.add_patch(Rectangle((j, 2 - i), 1, 1, facecolor=color,␣
 ↪edgecolor='black'))

legend_ax.set_xticks([0.5, 1.5, 2.5])
legend_ax.set_yticks([0.5, 1.5, 2.5])
legend_ax.set_xticklabels(["Low", "Med", "High"])
legend_ax.set_yticklabels(["High", "Med", "Low"])
legend_ax.set_xlabel("Home Value Appreciation")
legend_ax.set_ylabel("Treatment Size")
legend_ax.set_title("Legend", fontsize=12)
legend_ax.set_xlim(0, 3)
legend_ax.set_ylim(0, 3)
legend_ax.invert_yaxis()

plt.show()
```



This bivariate chloropleth reinforces the fact that there seems to be a relationship between the change in minority move-in share and home-value appreciation. The magenta represents areas with high minority move-ins and low home value appreciation. The teal represents areas with low minority move ins and high home value appreciation. The grey-blue represents moderate values on both. Most areas in the suburbs are either magenta, light pink, or teal, suggesting an inverse relation between change in minority move-ins and home value appreciation.

2.3 Regressions

[38]:
```
import statsmodels.api as sm
from statsmodels.iolib.summary2 import summary_col
from stargazer.stargazer import Stargazer
from IPython.core.display import HTML
```

2.3.1 Baseline models

[39]:
```
reg_data = data_table.dropna(subset=['treatment_size', 'value_diff_2019',␣
 ↪'msa_avg_value_ratio_2019'])
reg_data['white_x_treatment_size'] = reg_data['white'] *␣
 ↪reg_data['treatment_size']
reg_data['intercept'] = 1
X0 = reg_data[['intercept', 'treatment_size']] # Barebones regression
X1 = reg_data[['intercept', 'treatment_size', 'msa_avg_value_ratio_2019']] #␣
 ↪Baseline regression
X2 = reg_data[['intercept', 'treatment_size', 'msa_avg_value_ratio_2019',␣
 ↪'total_pop', 'white']] # Previous racial demos
X3 = reg_data[['intercept', 'treatment_size', 'msa_avg_value_ratio_2019',␣
 ↪'white_x_treatment_size']] # Interaction term
y = reg_data['value_ratio_2019']
model0 = sm.OLS(y, X0).fit()
model1 = sm.OLS(y, X1).fit()
model2 = sm.OLS(y, X2).fit()
model3 = sm.OLS(y, X3).fit()
```

[40]:
```
stargazer = Stargazer([model0, model1, model2, model3])
stargazer.custom_columns(["Change in Average Home Value (%) (2010-2017)"],[4])
HTML(stargazer.render_html())
# print(stargazer.render_latex()) #if you use Latex (Overleaf.com)
```

	Dependent variable: value ratio 2019			
	Change in Average Home Value (%) (2010-2019)			
	(0)	(1)	(2)	(3)
intercept	1.450***	0.040***	0.108***	0.043***
	(0.003)	(0.010)	(0.010)	(0.010)
MSA avg appreciation		0.963***	0.924***	0.961***
		(0.007)	(0.007)	(0.007)
total pop			0.000***	
			(0.000)	
treatment size	-0.324***	-0.263***	-0.155***	-0.158***
	(0.034)	(0.023)	(0.022)	(0.031)
white			-0.000***	
			(0.000)	
white×treatment size				-0.000***
				(0.000)
Observations	15919	15919	15919	15919
R^2	0.006	0.550	0.579	0.550
Adjusted R^2	0.006	0.550	0.579	0.550
Note:		*p<0.1; **p<0.05; ***p<0.01		

Model (0) shows a barebones regression with no controls – it indicates that there is a significant negative relationship between an increase in the share of minority move-ins and home value appreciation, but it's practically useless as it has very low explanatory power. Model (1) shows a baseline model controlling only for average MSA appreciation – it indicates there is still a negative relationship between minority move-ins and home value appreication, though it is smaller, and the model has much higher explanatory power. Model (2) indicates a similarly low positive relationship, and shows that there is little-to-no confounding occuring due to population size or whiteness of the area. Model (3) analyzes the interaction between zip-code whiteness and the change from the increase in minority move-in share and finds that there is no indication that whiteness affects the treatment effect.

2.3.2 Neighborhood income characteristics

[41]:
```
reg_data = data_table.dropna(subset=['treatment_size', 'value_diff_2019',␣
↪'msa_avg_value_ratio_2019', 'per_cap_inc', 'gini_index'])
reg_data['high_gini'] = (reg_data['gini_index'] > reg_data['gini_index'].
↪quantile(0.5)).astype(int)
reg_data['high_gini_x_treatment_size'] = reg_data['high_gini'] *␣
↪reg_data['treatment_size']
reg_data['intercept'] = 1
X4 = reg_data[['intercept', 'treatment_size', 'per_cap_inc',␣
↪'msa_avg_value_ratio_2019']]
```

```
X5 = reg_data[['intercept', 'treatment_size', 'gini_index',␣
 ↪'msa_avg_value_ratio_2019']]
X6 = reg_data[['intercept', 'treatment_size', 'gini_index',␣
 ↪'msa_avg_value_ratio_2019', 'high_gini_x_treatment_size']]
y = reg_data['value_ratio_2019']
model4 = sm.OLS(y, X4).fit()
model5 = sm.OLS(y, X5).fit()
model6 = sm.OLS(y, X6).fit()
```

[42]:
```
stargazer = Stargazer([model4, model5, model6])
stargazer.custom_columns(["Home Value Change (%) (2010-2017)"],[3])
HTML(stargazer.render_html())
# print(stargazer.render_latex()) #if you use Latex (Overleaf.com)
```

	Dependent variable: value ratio 2019		
	Home Value Change (%) (2010-2019)		
	(4)	(5)	(6)
Gini index		-0.013	-0.015
		(0.032)	(0.032)
High Gini×MMI-share change			-0.045
			(0.047)
intercept	0.126***	0.045***	0.046***
	(0.010)	(0.017)	(0.017)
MSA avg appreciation	1.006***	0.964***	0.964***
	(0.007)	(0.007)	(0.007)
Income (per capita)	-0.000***		
	(0.000)		
treatment size	-0.153***	-0.263***	-0.235***
	(0.022)	(0.023)	(0.037)
Observations	15919	15919	15919
R^2	0.583	0.550	0.550
Adjusted R^2	0.583	0.550	0.550
Note:	*p<0.1; **p<0.05; ***p<0.01		

Model (4) controls for income, but it does not seem that income has an economicially significant effect. It decreases the coefficient on treatment size (a 1% increase in the minority move-in proportion is associated with a 0.153% decrease in home-value appreciation), similarly to controlling for whiteness (this make sense, as due to socioeconomic factors, income and whitness are linked). (5) and (6) analyze how the effect varies over different levels of inequality in neighborhoods. The literature would suggest that those in neighborhoods with a lower Gini index would likely have preferences leading to a higher tolerance of neighborhood diversity, which would lead to a smaller change in home values after a change in MMI share, but the interaciton term in (6) does not reach significance, and neither does the control term in (5).

2.3.3 Political affiliation

[43]:
```
reg_data = data_table.dropna(subset=['treatment_size', 'value_diff_2019',␣
↪'msa_avg_value_ratio_2019', 'pct_rep'])
reg_data['rep_x_early_late'] = reg_data['pct_rep'] * reg_data['treatment_size']
reg_data['highly_rep'] = (reg_data['pct_rep'] > reg_data['pct_rep'].quantile(0.
↪5)).astype(int)
reg_data['high_rep_x_chg'] = reg_data['highly_rep'] *␣
↪reg_data['change_in_min_share']
reg_data['intercept'] = 1
X7 = reg_data[['intercept', 'treatment_size', 'pct_rep',␣
↪'msa_avg_value_ratio_2019']]
X8 = reg_data[['intercept', 'treatment_size', 'rep_x_early_late',␣
↪'msa_avg_value_ratio_2019', 'pct_rep']]
X9 = reg_data[['intercept', 'treatment_size', 'highly_rep', 'high_rep_x_chg',␣
↪'msa_avg_value_ratio_2019']]
y = reg_data['value_ratio_2019']
model7 = sm.OLS(y, X7).fit()
model8 = sm.OLS(y, X9).fit()
```

[44]:
```
stargazer = Stargazer([model7, model8])
stargazer.custom_columns(["Home Value Change (%) (2010-2019)"],[2])
HTML(stargazer.render_html())
# print(stargazer.render_latex()) #if you use Latex (Overleaf.com)
```

	Dependent variable: value ratio 2019	
	Home Value Change (%) (2010-2019)	
	(7)	(8)
Highly Republican × Treatment size		0.227***
		(0.025)
Highly republican		0.009**
		(0.004)
intercept	0.048***	0.047***
	(0.013)	(0.011)
MSA average appreciation	0.962***	0.957***
	(0.007)	(0.007)
Percent republican	-0.013	
	(0.012)	
treatment size	-0.262***	-0.304***
	(0.023)	(0.023)
Observations	15889	15889
R^2	0.549	0.552
Adjusted R^2	0.549	0.552
Note:		*p<0.1; **p<0.05; ***p<0.01

Model (7) controls for republican voting behavior, but finds there is not a statistically significant relationship between republicanness and home value appreciation over this time. (8) uses similar intuition to the inequality interaction regression. One may assume that more republican geographies will have a higher chance of having more discriminatory preferences, which would suggest an interaction term with a more negative coefficient. The regression actually finds the opposite, that in more republican geographies a 1% increase in minority move-ins is associated with a 0.227% increase in home value appreciation (as opposed to a 0.304% decrease in general).

2.3.4 Preferred specification

When evaluating regressions, I looked for significant terms. An ideal regression has clear conclusions with significant terms and a high R^2.

Many regression specifications above introduce control variables that end up having low significance, and many (surprisingly, at times) appear to have no clear impact on the outcome variable. As a result, the regression that I find tells the best story is the regression that analyzes the interaction between republican share and treatment size:

$$\begin{aligned}\text{Home value appreciation} = \beta_0 &+ \beta_1(\text{Increase in minority move-in share}) \\ &+ \beta_2(\text{Home value appreciation of MSA}) \\ &+ \beta_3(\text{Highly Republican}) \\ &+ \beta_4(\text{Highly Republican} \times \text{Increase in minority move-in share})\end{aligned}$$

This model indicates that there is a negative, significant effect from an increased minority move-in share on home-value appreciation. Surprisingly, it also indicates that this effect works in the opposite direction in highly republican geographies.

3 Final Project

3.1 Potential Data to Scrape

My project assesses home value appreciation, and uses the Zillow ZHVI as its data source for home-values. ZHVI is great, but it only provides data on a ZIP-code level, and is missing data for many ZIP codes. This is less than ideal for two reasons: Firstly, ZIP codes are quite large, but also have incredibly varying population sizes. This causes the geographic precision of our analysis to not be well defined. Secondly, having randomly missing data is obviously unfortunate, and so being able to rectify that would be wonderful.

My ideal web-scraped data would be from either Zillow or Realtor.com or another MLS aggregator. I would want the home listings (and potentially rental listings) from 2012-2019. I could then aggregate these by census tract to create my own version of a "ZHVI" that has finer and more available data. This would increase the precision of my analysis, and remove some merging friction (the loan data is already on census tract level).

3.2 Potential Challenges

For this data, it would be useful to run the program every month or so from 2012-2019. This would allow me to know about most of the listings and sales. Unfortunately, this project began in 2025, so we need archival websites, which introduces a lot of friction. Archive websites frequently have better webscrape prevention, which can slow down scraping. On top of this, archive websites are incredibly slow to load. Because of this, I will restrict to only homes in the city of Boston, MA. Boston is a good candidate city because we have a lot of data on minority move-ins, it has a lot of single-family neighborhoods, and it's "city proper" is quite large. In addition, due to a lack of archival web availability, I can only really look at the change from 2014 to present, meaning that we are scraping solely for mid-2014 and April 2025. This will reduce accuracy, but significantly speed up computation, and serve as a "proof-of-concept" that this webscraping technique could work if ran consistently.

In addition, I find that there are data availability issues due to each time simply being one snapshot, so I only get about 400 listings. This means I don't have sufficient data to feel comfortable aggregating to something smaller than ZIP code, such as census tract.

3.3 Scraping

[45]:
```
import requests
from bs4 import BeautifulSoup
import time
import random
import os
import re
```

To begin, we first setup our user agents and webscraping parameters. I found that both Web Archive and Realtor/Redfin have reasonably good webscrapre prevention, but that a good workaround is using several different user agents to obscure traffic.

[46]:
```
user_agents = [
    "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like␣
 ↪Gecko) Chrome/58.0.3029.110 Safari/537.3",
    "Mozilla/5.0 (Windows NT 6.1; WOW64; rv:35.0) Gecko/20100101 Firefox/35.0",
    "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like␣
 ↪Gecko) Chrome/80.0.3987.122 Safari/537.36",
    "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML,␣
 ↪like Gecko) Chrome/91.0.4472.114 Safari/537.36"
]

checkpoint_file = "scraped_listings.csv"
start_page = 1
max_empty_pages = 5
empty_page_count = 0
all_listings = []
```

3.3.1 2014

Next, we define an `extract_listing_data` function that can be given page details and convert to listing data.

[47]:
```python
def extract_listing_data(li_tags):
    listings = []
    current = {}

    for li in li_tags:
        text = li.get_text(strip=True)

        # Extract address
        address = li.select_one('.listing-street-address')
        if address:
            current['address'] = address.get_text(strip=True)

        # Extract city, state, and postal code
        city_state_zip = li.select_one('.listing-city, .listing-region, .
↪listing-postal')
        if city_state_zip:
            current['citystatezip'] = ' '.join([item.get_text(strip=True) for␣
↪item in city_state_zip.find_all(['span'])])

        # Extract price
        price = li.select_one('.listing-price')
        if price:
            current['price'] = price.get_text(strip=True)

        # Extract beds and baths
        beds_baths = li.select_one('.listing-beds')
        if beds_baths:
            beds_baths_text = beds_baths.get_text(strip=True)
            if 'Bd' in beds_baths_text and 'Ba' in beds_baths_text:
                current['beds'], current['baths'] = beds_baths_text.
↪split('Ba')[0].strip(), beds_baths_text.split('Ba')[1].strip()

        # Extract square footage
        sqft = li.select_one('.listing-sqft')
        if sqft:
            current['sqft'] = sqft.get_text(strip=True)

        # If we have a complete listing (address and price), store it
        if 'address' in current and 'price' in current:
            listings.append(current)
            current = {}  # Reset for next listing

    return listings
```

Next, we actually scrape. This code begins by looking for a "checkpoint file" so that if I run into issues while scraping, such as being denied due to too many requests or bot-like behavior, I can pick up where I left off. Next, it parses through each page. Unfortunately, Web Archive is missing some pages due to bot denial that they experienced, so I have the code continue to look through pages until it has five empty pages in a row.

The code begins by sending a request using a random user agent (one of the ones defined above), then checks if it was successful, and then parses through to find all listings. It then uses the above function to extract the data from those listings, and saves it before progressing to the next page.

[48]:
```
# Load checkpoint data if exists
if os.path.exists(checkpoint_file):
    print(f"Checkpoint found: resuming from {checkpoint_file}")
    all_listings = pd.read_csv(checkpoint_file).to_dict(orient='records')

while empty_page_count < max_empty_pages:
    url = f"https://web.archive.org/web/20140401113539/http://www.realtor.com/
↪realestateandhomes-search/Boston_MA/pg-{start_page}"
    print(f"Scraping page {start_page}...")

    try:
        response = requests.get(url, headers={"User-Agent": random.
↪choice(user_agents)})

        if response.status_code != 200:
            print(f"Page {start_page} returned {response.status_code}.")
            empty_page_count += 1
            start_page += 1
            continue

        soup = BeautifulSoup(response.text, 'html.parser')
        li_tags = soup.select("li[class*=listing]")

        if not li_tags:
            print(f"No listings found on page {start_page}.")
            empty_page_count += 1
            start_page += 1
            continue

        listings = extract_listing_data(li_tags)

        if listings:
            all_listings.extend(listings)
            empty_page_count = 0  # Reset counter on success
        else:
            empty_page_count += 1

        # Save checkpoint every successful iteration
        pd.DataFrame(all_listings).to_csv(checkpoint_file, index=False)

    except requests.exceptions.RequestException as e:
```

```
        print(f"Request error on page {start_page}: {e}")
        break
    except ConnectionRefusedError:
        print(f"Connection refused error on page {start_page}. Saving progress␣
↪and stopping.")
        # Save checkpoint and stop
        pd.DataFrame(all_listings).to_csv(checkpoint_file, index=False)
        break

    start_page += 1
    time.sleep(5)

# Final output
print(f"\nTotal listings scraped: {len(all_listings)}")
for i, listing in enumerate(all_listings, 1):
    print(f"{i}: {listing}")
```

[49]:
```
scraped_listings = pd.read_csv('data/scraped_listings.csv')
```

Next, we clean the addresses of the listings to improve the chance of finding the ZIP code.

[50]:
```
def clean_address(address):
    """
    Cleans the address by removing unit numbers (e.g., Apt, Unit) and adding␣
↪city/state if missing.
    """
    # Remove unit numbers and anything after #
    address = re.sub(r'(\s*#\s*\d+|\s*(Apt|Unit|Ste|Suite)\s*\w+)', '', address)

    # Add "Boston, MA" if no city/state is included
    if not re.search(r'\b(Boston|Cambridge|Brookline)\b', address):
        address += ', Boston, MA'

    return address.strip()
```

Now, we use Google's geolocating API to find the ZIP code associated with the address in the listing.

[51]:
```
from geopy.geocoders import GoogleV3

def get_zip_code(address):
    API_KEY = ''
    geolocator = GoogleV3(api_key=API_KEY)
    """
    Uses Google Maps API to get the zip code of the address.
    Returns the zip code or None if not found.
    """
    try:
```

```
        location = geolocator.geocode(address)
        if location:
            # Get latitude and longitude of the address
            lat, lon = location.latitude, location.longitude

            # Use reverse geocoding with latitude and longitude
            address_details = geolocator.reverse((lat, lon), language='en',␣
↪exactly_one=True)
            if address_details:
                # Attempt to extract zip code from address details
                for component in address_details.raw.get('address_components',␣
↪[]):
                    if 'postal_code' in component['types']:
                        return component['long_name']
        return None
    except Exception as e:
        print(f"Error getting zip code for {address}: {e}")
        return None

# Assuming 'all_listings' is your DataFrame
df = pd.read_csv('scraped_listings.csv')  # Load the DataFrame with your scraped␣
↪data

# Add a column for zip code
df['zip_code'] = None

# Loop through the rows of the DataFrame and get the zip code
for index, row in df.iterrows():
    address = row['address']
    if address:
        cleaned_address = clean_address(address)  # Clean the address first
        print(f"Getting zip code for {cleaned_address}...")
        zip_code = get_zip_code(cleaned_address)
        df.at[index, 'zip_code'] = zip_code
    time.sleep(1)  # Sleep to avoid hitting API too frequently

# Save the updated DataFrame with zip code information
df.to_csv('scraped_listings_with_zip.csv', index=False)

print(f"Finished adding zip codes. Total listings: {len(df)}")
```

[52]:
```
scraped_zip = pd.read_csv('data/scraped_listings_with_zip.csv')
scraped_zip['price'] = scraped_zip['price'].str.replace(r'[\$,]', '', regex=True)
scraped_zip['price'] = pd.to_numeric(scraped_zip['price'], errors='coerce')
```

Finally, we group by zip code and ensure the zip code format is correct.

[53]:
```
scrape = scraped_zip.groupby('zip_code').agg({
    'price': 'mean',
    'sqft' : 'count'
}).reset_index()
scrape['zip_code'] = scrape['zip_code'].astype(str).str.zfill(5)
```

3.3.2 2025

For the live data, we use Redfin, which has a completely different web format. As a result, a new `extract_listing_data` helper function is needed.

[54]:
```
def extract_listing_data(div_blocks):
    listings = []

    for div in div_blocks:
        try:
            price_tag = div.select_one(".bp-Homecard__Price--value")
            address_tag = div.select_one(".bp-Homecard__Address")

            if not price_tag or not address_tag:
                continue  # skip if either is missing

            price = price_tag.get_text(strip=True)
            address = address_tag.get_text(strip=True)

            listing = {
                "price": price,
                "address": address
            }

            # Optional: extract beds, baths, sqft if available
            beds_tag = div.select_one(".bp-Homecard__Stats--beds")
            baths_tag = div.select_one(".bp-Homecard__Stats--baths")
            sqft_tag = div.select_one(".bp-Homecard__LockedStat--value")

            if beds_tag:
                listing["beds"] = beds_tag.get_text(strip=True)
            if baths_tag:
                listing["baths"] = baths_tag.get_text(strip=True)
            if sqft_tag:
                listing["sqft"] = sqft_tag.get_text(strip=True)

            listings.append(listing)

        except Exception as e:
            print(f"Error parsing listing: {e}")
```

```
    return listings
```

[55]:
```
checkpoint_file = "2019_scraped_listings.csv"

# Resume from checkpoint if it exists
if os.path.exists(checkpoint_file):
    print(f"Checkpoint found: resuming from {checkpoint_file}")
    all_listings = pd.read_csv(checkpoint_file).to_dict(orient='records')

# Start scraping
while empty_page_count < max_empty_pages:
    url = f"https://www.redfin.com/city/1826/MA/Boston/page-{start_page}"
    print(f"\nScraping page {start_page}...")

    try:
        response = requests.get(url, headers={"User-Agent": random.
 ↪choice(user_agents)})

        if response.status_code != 200:
            print(f"Page {start_page} returned status {response.status_code}.")
            empty_page_count += 1
            start_page += 1
            time.sleep(5)
            continue

        soup = BeautifulSoup(response.text, 'html.parser')
        data_divs = soup.find_all("div", class_="bp-Homecard__Content")

        if not data_divs:
            print(f"No listings found on page {start_page}.")
            empty_page_count += 1
            start_page += 1
            time.sleep(5)
            continue

        page_listings = []

        for div in data_divs:
            try:
                price_tag = div.select_one(".bp-Homecard__Price--value")
                address_tag = div.select_one(".bp-Homecard__Address")

                if not price_tag or not address_tag:
                    continue

                price = price_tag.get_text(strip=True)
                address = address_tag.get_text(strip=True)
```

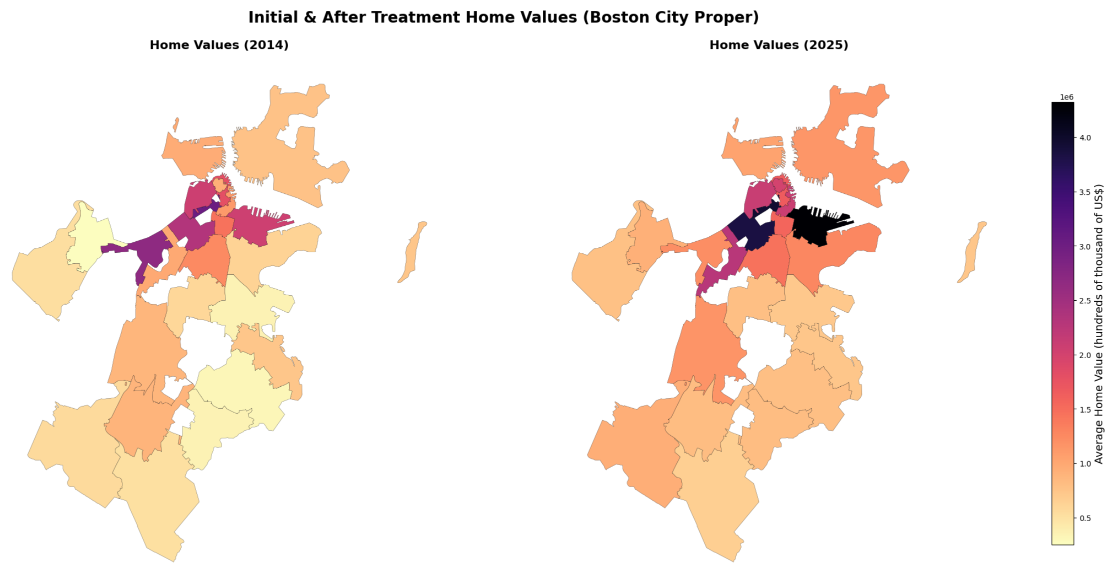
```
                page_listings.append({"address": address, "price": price})
            except Exception as e:
                print(f"Error parsing listing: {e}")

        if page_listings:
            all_listings.extend(page_listings)
            empty_page_count = 0
            pd.DataFrame(all_listings).to_csv(checkpoint_file, index=False)
            print(f"Page {start_page} listings added: {len(page_listings)}")
        else:
            print(f"No valid listings on page {start_page}.")
            empty_page_count += 1

    except requests.exceptions.RequestException as e:
        print(f"Request error on page {start_page}: {e}")
        break
    except ConnectionRefusedError:
        print(f"Connection refused on page {start_page}. Saving and stopping.")
        pd.DataFrame(all_listings).to_csv(checkpoint_file, index=False)
        break

    start_page += 1
    time.sleep(5)

# Final output
print(f"\nTotal listings scraped: {len(all_listings)}")
for i, listing in enumerate(all_listings, 1):
    print(f"{i}: {listing}")
```

3.4 Webscraped Vizualizations

[56]:
```
scrape_19 = pd.read_csv('data/2019_scraped_listings.csv')
scrape_19['zip'] = scrape_19['address'].str.extract(r'(\d{5})')
scrape_19['price_19'] = pd.to_numeric(scrape_19['price'].str.replace(r'[$,]',␣
 ↪'', regex=True), errors='coerce')
scrape_19 = scrape_19.groupby('zip').agg({'price_19': 'mean', 'address':␣
 ↪'count'}).reset_index()
scrape_19.rename(columns={'address': 'listing_count_19'}, inplace=True)

scraped = scrape.merge(scrape_19, left_on='zip_code', right_on="zip",␣
 ↪how='left').dropna(subset=['price_19'])
```

[57]:
```
zips = gpd.read_file('shapefiles/zips/US_zcta_2010.shp')
zips = zips.merge(scraped, left_on='GEOID10', right_on='zip_code', how='inner')

vmin = min(scraped['price'].min(), scraped['price_19'].min())
```

```python
vmax = max(scraped['price'].max(), scraped['price_19'].max())

fig, (gax1, gax2) = plt.subplots(1, 2, figsize=(22, 10), constrained_layout=True)

# First plot
cmap1 = "magma_r"
plot1 = zips.plot(
    ax=gax1, edgecolor='black', linewidth=0.2, cmap=cmap1,
    column='price', legend=False,  # Set legend=False here
    vmin=vmin, vmax=vmax  # Set the color scale to match the common range
)
gax1.set_title('Minority Move-In Share (initial period)', fontsize=16,␣
 ↪fontweight='bold', pad=15)
gax1.axis('off')
gax1.set_aspect('equal')  # Force equal aspect ratio

# Second plot
plot2 = zips.plot(
    ax=gax2, edgecolor='black', linewidth=0.2, cmap=cmap1,
    column='price_19', legend=False,
    vmin=vmin, vmax=vmax  # Set the color scale to match the common range
)
gax2.set_title('Minority Move-In Share (treatment period)', fontsize=16,␣
 ↪fontweight='bold', pad=15)
gax2.axis('off')
gax2.set_aspect('equal')  # Force equal aspect ratio

# Add a single shared colorbar
sm = plt.cm.ScalarMappable(cmap='magma_r', norm=plt.Normalize(vmin=vmin,␣
 ↪vmax=vmax))
sm.set_array([])  # Only needed for the colorbar to show properly
cbar = fig.colorbar(sm, ax=[gax1, gax2], orientation='vertical', fraction=0.02,␣
 ↪pad=0.04)
cbar.set_label('Minority Move-In Share', fontsize=14)

# Add the overall title
fig.suptitle("Initial & After Treatment Home Values (Boston City Proper)",␣
 ↪fontsize=20, fontweight='bold', y=1.05)

# Adjust plot spacing
fig.subplots_adjust(wspace=0.3)

plt.show()
```

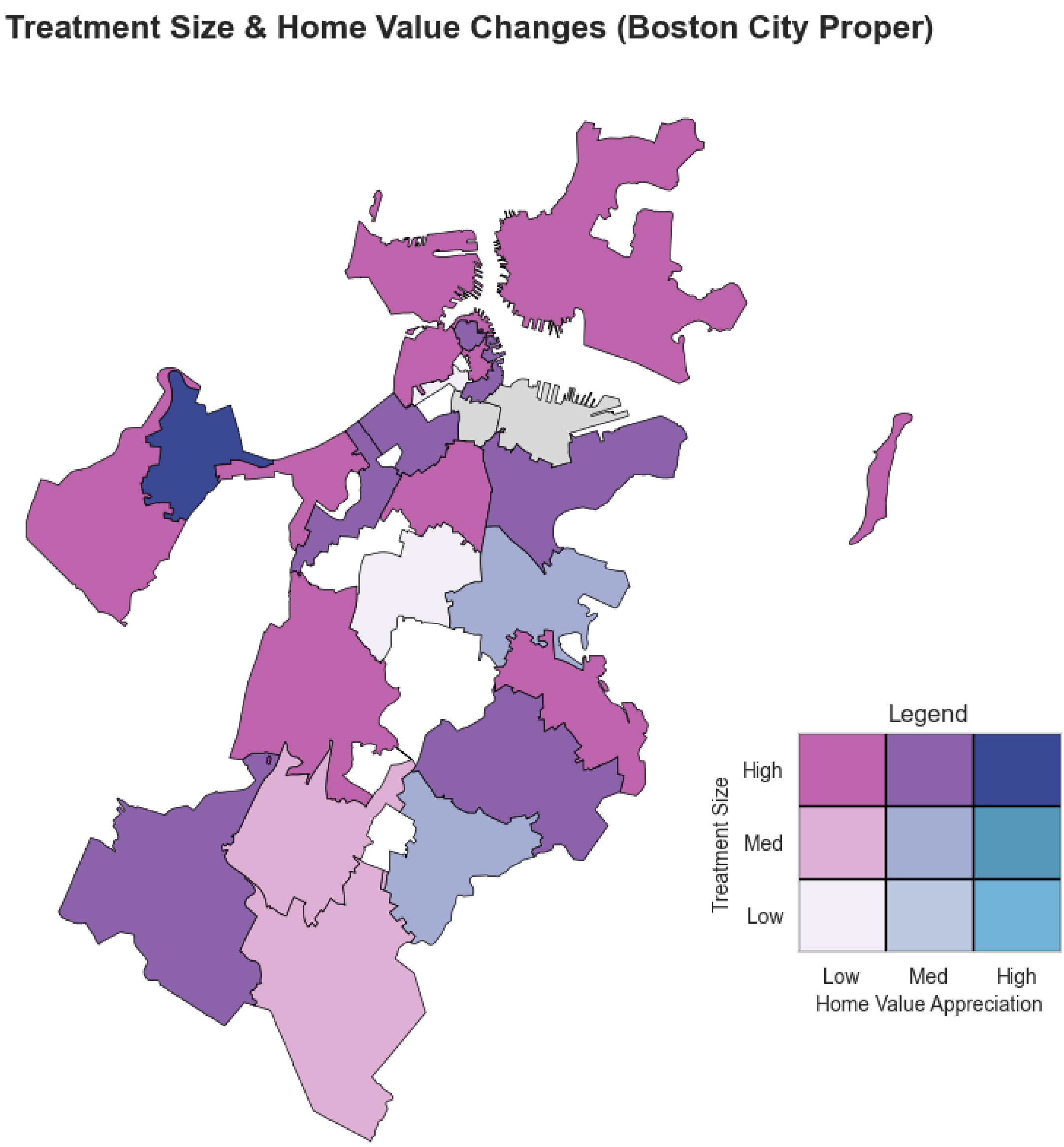


This map shows the set of data gathered from webscraping. Because we were scaled back to only Boston City Proper, you can see that there is significant home-value increases across the board. Unfortunately, it appears that these two times of webscraping weren't enough to remove data gaps (in fact, we have more). However, it is clear that if this script could more easily gain archival data, we would be able to fill in gaps.

[58]:
```python
def normalize(series):
    return (series - series.min()) / (series.max() - series.min())

zips['GEOID10'] = zips['GEOID10'].astype(float)
zips = zips.merge(data_table, left_on='GEOID10', right_on='zip', how='inner')
df = zips
df['price_diff'] = df['price_19'] / df['price']
df['norm_movein'] = normalize(df['treatment_size'])
df['norm_value'] = normalize(df['price_diff'])
```

[59]:
```python
bivariate_colormap = [
    ['#f1eef6', '#bdc9e1', '#73b3d8'],  # Low move-in
    ['#dfb0d6', '#a5add3', '#5698b9'],  # Medium move-in
    ['#be64ac', '#8c62aa', '#3b4994'],  # High move-in
]

def bivariate_color(movein, value):
    if np.isnan(movein) or np.isnan(value):
        return '#d9d9d9'  # fallback for NaNs
    row = min(2, int(movein * 2.999))
    col = min(2, int(value * 2.999))
    return bivariate_colormap[row][col]
```

```
# Apply color
df['color'] = df.apply(lambda row: bivariate_color(row['norm_movein'],␣
↪row['norm_value']), axis=1)

# Plot
fig, ax = plt.subplots(figsize=(12, 10))
df.plot(ax=ax, color=df['color'], edgecolor='black', linewidth=0.5,
        missing_kwds={'color': 'lightgray'})

# Title
ax.set_title("Treatment Size & Home Value Changes (Boston City Proper)",
             fontsize=16, fontweight='bold', pad=15)
ax.axis('off')

# Legend
fig.subplots_adjust(right=1.0)
legend_size = 0.15
legend_ax = fig.add_axes([0.75, 0.35 - legend_size / 2, legend_size,␣
↪legend_size])

for i in range(3):
    for j in range(3):
        color = bivariate_colormap[i][j]
        legend_ax.add_patch(Rectangle((j, 2 - i), 1, 1, facecolor=color,␣
↪edgecolor='black'))

legend_ax.set_xticks([0.5, 1.5, 2.5])
legend_ax.set_yticks([0.5, 1.5, 2.5])
legend_ax.set_xticklabels(["Low", "Med", "High"])
legend_ax.set_yticklabels(["High", "Med", "Low"])
legend_ax.set_xlabel("Home Value Appreciation")
legend_ax.set_ylabel("Treatment Size")
legend_ax.set_title("Legend", fontsize=12)
legend_ax.set_xlim(0, 3)
legend_ax.set_ylim(0, 3)
legend_ax.invert_yaxis()

plt.show()
```



This map shows less certainty around the relationship between treatment size and home value changes. This is likely due to the fact that we are using a small sample size of data, and that the "post-treatment" home values are after so much time. However, it does show that there is a relationship between treatment size and home value changes, which is consistent with the above regressions, maps, and plots.

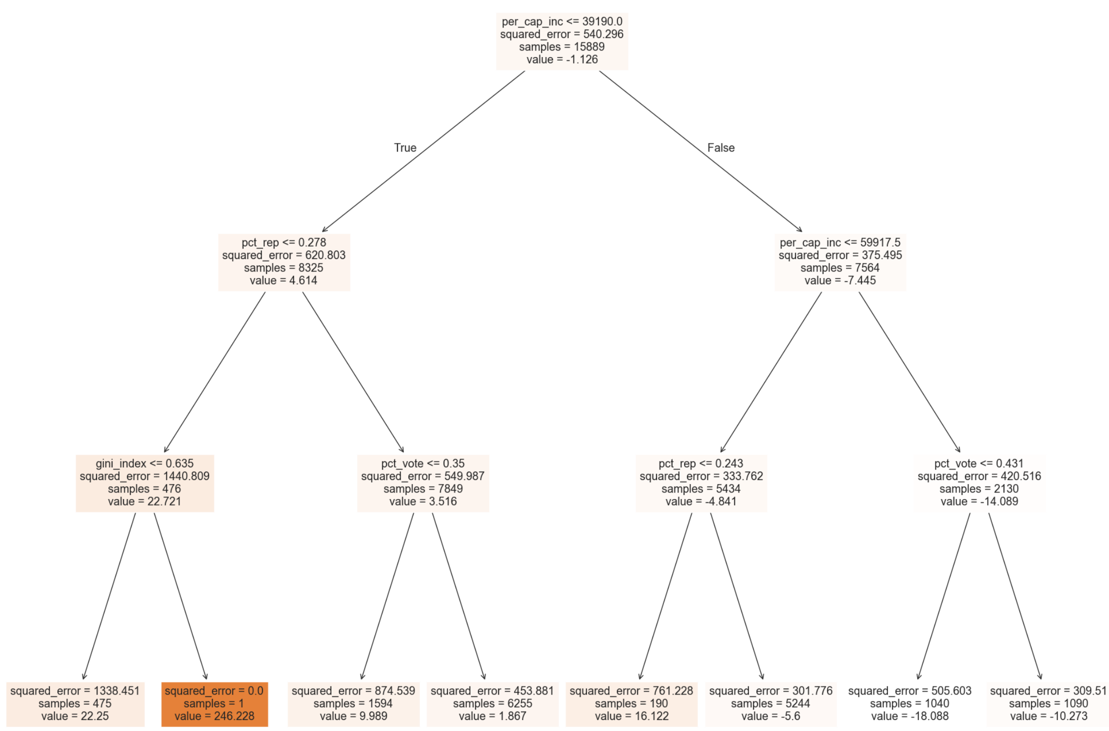
3.5 Regression Trees

[60]:
```
from sklearn import tree
from sklearn.metrics import mean_squared_error,confusion_matrix,␣
↪classification_report
```

[61]:
```
data_ml_full = data_table[['treatment_size', 'value_diff_2019', 'pct_rep',␣
↪'late_move_ins', 'white', 'pct_dem', 'pct_vote', 'black', 'gini_index',␣
↪'per_cap_inc']].dropna()
X_f = data_ml_full.drop(columns=['value_diff_2019'])
for col in list(X_f):
    X_f[col] = X_f[col].astype(float)
y_f = data_ml_full['value_diff_2019'] * 100
sqft_tree = tree.DecisionTreeRegressor(max_depth=3).fit(X_f,y_f)
y_pred = sqft_tree.predict(X_f)
print("MSE:", mean_squared_error(y_f, y_pred))
```

```
MSE: 469.4532489841935
```

[62]:
```
sqrf_fig = plt.figure(figsize=(25,20))
sqrf_fig = tree.plot_tree(sqft_tree, feature_names=X_f.columns, filled=True)
```

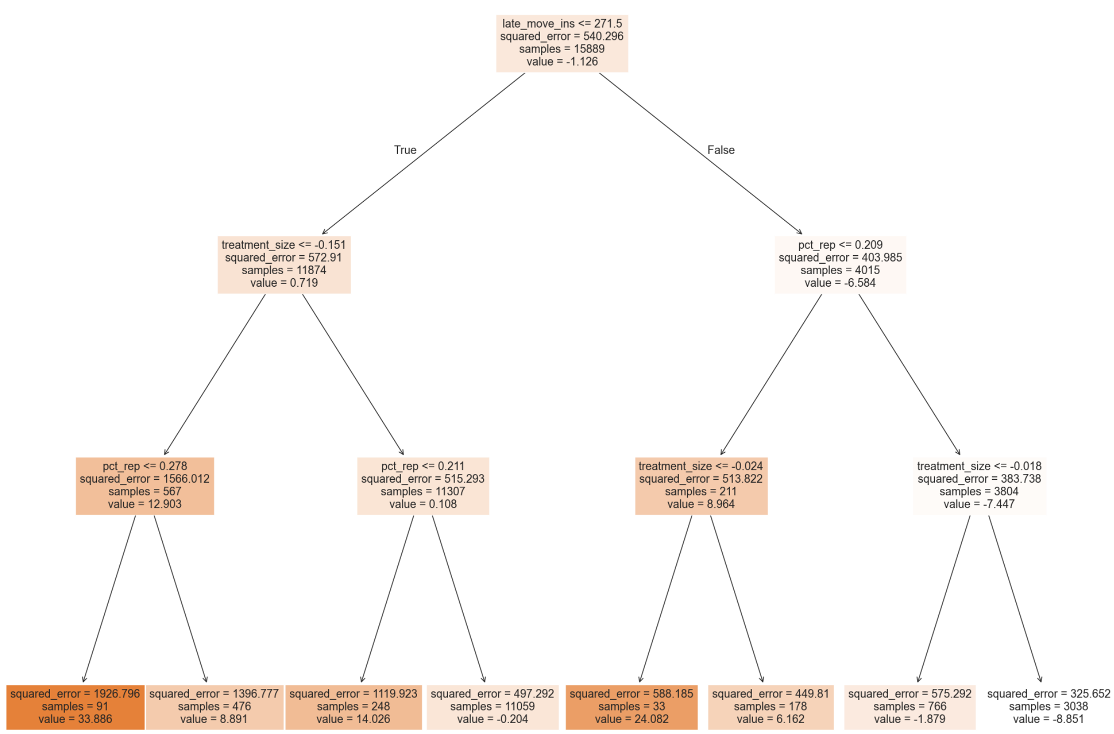


This tree with all of my X variables indicates that the income level, income inequality, and political activity are the largest predictors of home value appreciation relative to metropolitan area. Having a higher per capita income level suggests lower home value appreciation, as does being more highly republican. Higher political activity is associated with higher home value appreciation in higher-income areas, but lower home value appreciation in lower-income areas. All terms have an incredibly high level of error.

[63]:
```
data_ml_sub = data_table[['treatment_size', 'value_diff_2019', 'pct_rep', 'late_move_ins', 'white']].dropna()
X_sub = data_ml_sub.drop(columns=['value_diff_2019'])
for col in list(X_sub):
    X_sub[col] = X_sub[col].astype(float)
y = data_ml_sub['value_diff_2019'] * 100
sqft_tree = tree.DecisionTreeRegressor(max_depth=3).fit(X_sub,y)
y_pred = sqft_tree.predict(X_sub)
print("MSE:", mean_squared_error(y, y_pred))
```

```
MSE: 512.7436461417611
```

[64]:
```
sqrf_fig = plt.figure(figsize=(25,20))
sqrf_fig = tree.plot_tree(sqft_tree, feature_names=X_sub.columns, filled=True)
```



This tree just contains my main X variables (change in minority move-ins, percent of population that voted republican, total treatment move-ins, and proportion of the zip code that is white). This tree seems to indicate that the total quantity of move-ins is of high importance, with fewer move-ins indicating higher appreciation (this is counterintuitive to economic intuition). In places with higher move-ins, the proportion of republicans is the next most important variable, with less republican areas experiencing higher home-value appreciation. Next is the treatment. ZIP Codes with a high quantity of move-ins that recieve a high treatment dose experience lower home-value appreciation than those who don't. High move-in republican ZIP codes with higher treatment doses have even lower home-value appreciations. In low move-in ZIP codes, treatment dose is the next most important, followed by republican proportion. The same effects are seen. The error is higher on this model due to the removal of many control variables.

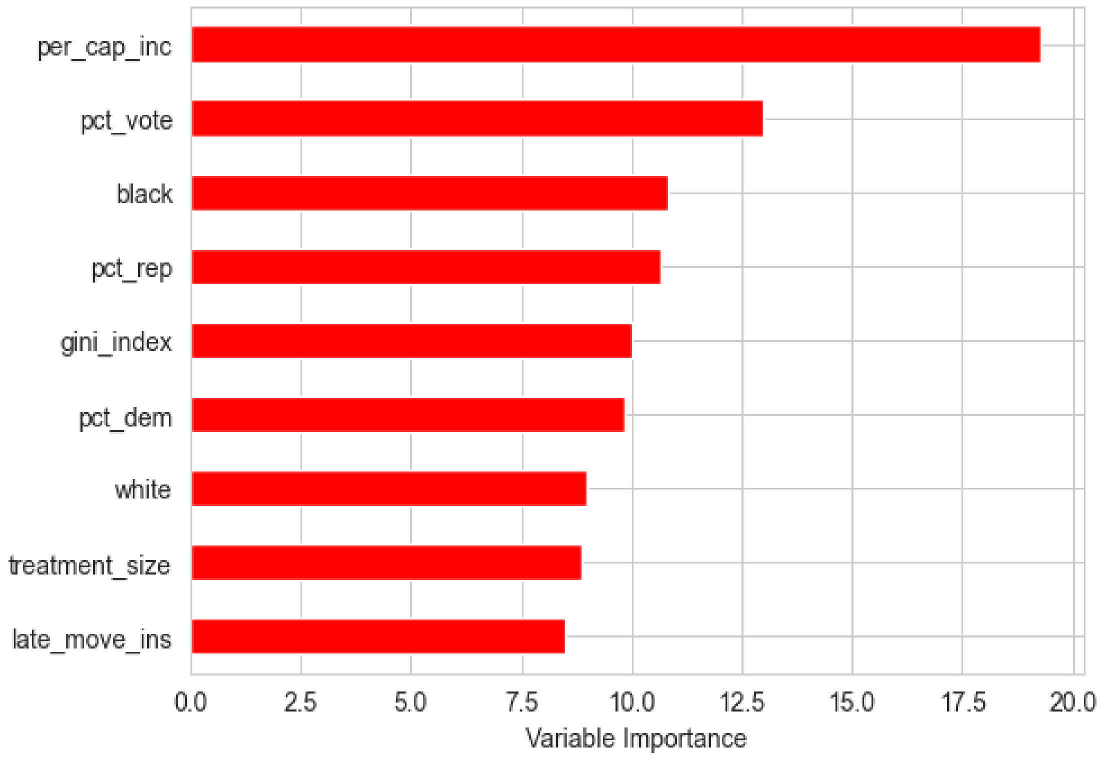
3.6 Random Forest

[65]:
```
from sklearn.ensemble import RandomForestRegressor
```

[66]:
```
regr_RF = RandomForestRegressor(max_features=5, random_state=1).fit(X_f, y_f)
pred = regr_RF.predict(X_f)
mean_squared_error(y_f, pred)
```

[66]:
```
49.636918727897275
```

[67]:
```
Importance = pd.DataFrame({'Importance':regr_RF.feature_importances_*100},␣
 ↪index=X_f.columns)
Importance.sort_values('Importance', axis=0, ascending=True).plot(kind='barh',␣
 ↪color='r', )
plt.xlabel('Variable Importance')
plt.gca().legend_ = None
```



This bar chart indicates that by far the most important variable is per capita income. This is followed by voting charactersitcs and neighborhood demographics, and then by the treatment size.

3.7 OLS vs ML

The results from my ML seem to run slightly contrary to our regressions, which indicated that income had little to no effect on home-value appreciation. It's possible that this is due to a slightly distinct y between our ML models and our regressions. In the regressions, we looked at the raw home value appreciation, and then added the average home-value appreciation in the MSA as a control term. In the ML models, to improve clarity, we are looking at the difference between home value appreciation and the average home value appreciation the MSA.

My regression trees (and random forest model) allow for significantly more analysis of interactions between variables.

3.8 Conclusion

In this paper, I analyze the link between an increase in the share of minority move-ins (a treatment) and home values on a neighborhood level. I construct a zip-code level dataset that includes, from Fannie Mae and Freddie Mac data, the share of minority move-ins in an initial period, and the share of minority move-ins in a treatment period. The data also includes Zillow data showing the change in home values. For much of the analysis, this change in home values is normalized to the average change in home values to the MSA the zip code is in.

My findings indicate a statistically significant negative relationship between an increase in minority move-ins and home-value appreication. This effect is present when normalizing for MSA-level appreciation, and is consistent with webscraped listing data. Interestingly, this relationship is significantly less negative in highly (top 50%) Republican geographies. This could be due to different preference maps, or some other endogenous factor.

More analysis is needed, including modfying the methodology to something more similar to an event study, to allow for more years of treatment and better consideration for treatment size. Presently, there is too much endogeneity for any causal analysis. It's possible that modifying the study structure could change that. In addition, it would be insightful to use the data and regressions to estimate values for the parameters in the rational discrimination theoretical model this analysis is based upon.

3.9 References

Appel, I., & Nickerson, J. (2016). Pockets of Poverty: The Long-Term Effects of Redlining. https://papers.ssrn.com/sol3/Delivery.cfm?abstractid=2852856

Arrow, K. (1971). *The Theory of Discrimination.*
https://dataspace.princeton.edu/handle/88435/dsp014t64gn18f

Avetian, V. (2022). Consider the Slavs: Overt Discrimination and Racial Disparities in Rental Housing. https://www.tse-fr.eu/sites/default/files/TSE/documents/conf/2022/echoppe/avetian.pdf

Badarinza, C., Balasubramaniam, V., & Ramadorai, T. (2024). In Search of the Matching Function in the Housing Market (SSRN Scholarly Paper No. 4594519). https://doi.org/10.2139/ssrn.4594519

Becker, G. S. (1971). The Economics of Discrimination. University of Chicago Press.

Combes, P.-P., Decreuse, B., Laouénan, M., & Trannoy, A. (2016). Customer Discrimination and Employment Outcomes: Theory and Evidence From the French Labor Market. *Journal of Labor Economics*, 34(1), 107–160. https://doi.org/10.1086/682332

Combes, P.-P., Decreuse, B., Schmutz, B., & Trannoy, A. (2018). Neighbor Discrimination Theory and Evidence from the French Rental Market. Journal of Urban Economics, 104, 104–123. https://doi.org/10.1016/j.jue.2018.01.002

Korver-Glenn, E. (2018). Compounding inequalities: How racial stereotypes and discrimination accumulate across the stages of housing exchange. American Sociological Review, 83(4), 627–656. https://doi.org/10.1177/0003122418781774

Manson, S., Schroeder, J., Van Riper, D., Knowles, K., Kugler, T., Roberts, F., & Ruggles, S. (2024). National historical geographic information system: Version 19. 0 [Dataset]. Minneapolis, MN: IPUMS. https://doi.org/10.18128/D050.V19.0

Quillian, L., Lee, J. J., & Honoré, B. (2020). Racial discrimination in the u. S. Housing and mortgage lending markets: A quantitative review of trends, 1976–2016. Race and Social Problems, 12(1), 13–28. https://doi.org/10.1007/s12552-019-09276-x

Rothstein, R. (2017). *The Color of law: A Forgotten History of How Our Government Segregated America* (First edition). Liveright Publishing Corporation, a division of W. W. Norton & Company.

Sood, A., Erhman-Solberg, K. (2024). The Long Shadow of Housing Discrimination: Evidence from Racial Covenants https://drive.google.com/file/d/1uLSaQxWiSHKMuckF2gFpATywQD2J7No5/

Appendix

A Feedback Incorporation

1. Fixed bugs in data import
2. Added subset table for zip codes with minority move-ins
3. Changed message plot to reflect new data and present a more interesting conclusion
4. Added boundaries to maps
5. Elected not to attempt aggregations for the nationwide map – once we are on a county level there are little-to-no insights to be made.
6. Modified bivariate chloropleth to have better colors
7. Improved map explanation
8. Added more regression coefficient interpretations, improved regression variable names.
9. You may notice that the maps, regressions, and plots look different. This is due to fixing a major bug in the data import.
10. Enhanced conclusion's analysis of results