Analysis Notebook 1

Hypothesis: The average income in a county significantly influences property prices in NY.

Understanding the factors that influence property prices, such as average income and property characteristics, is critically important from a social perspective. Housing affordability is a significant concern for many individuals and families, especially in regions with high living costs like New York. By analyzing and predicting property prices based on various economic and demographic variables, policymakers and urban planners can gain valuable insights into housing market dynamics. This knowledge enables them to develop more effective housing policies, address issues of affordability and inequality, and ensure that housing markets operate more fairly.

Data Preparation: I created a new dataset by merging data from three different sources:

1. U.S. Bureau of Economic Analysis (BEA) www.bea.gov: Provided average income by county.
2. World Population Review worldpopulationreview.com: Provided population data for 2022 by county.
3. Kaggle www.kaggle.com: Provided a dataset with New York house prices, including details such as broker titles, house types, prices, number of bedrooms and bathrooms, property square footage, addresses, state, administrative and local areas, street names, and geographical coordinates.

Comment: This is a cool idea !

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import csv
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_scor
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

%pip install seaborn --upgrade
```

```
Requirement already satisfied: seaborn in /srv/conda/lib/python3.11/site-pac
kages (0.13.2)
Requirement already satisfied: numpy!=1.24.0,>=1.20 in /srv/conda/lib/python
3.11/site-packages (from seaborn) (1.26.4)
Requirement already satisfied: pandas>=1.2 in /srv/conda/lib/python3.11/site
-packages (from seaborn) (2.2.2)
Requirement already satisfied: matplotlib!=3.6.1,>=3.4 in /srv/conda/lib/pyt
hon3.11/site-packages (from seaborn) (3.7.3)
Requirement already satisfied: contourpy>=1.0.1 in /srv/conda/lib/python3.1
1/site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (1.2.1)
Requirement already satisfied: cycler>=0.10 in /srv/conda/lib/python3.11/sit
e-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /srv/conda/lib/python3.1
1/site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (4.53.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /srv/conda/lib/python3.1
1/site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (1.4.5)
Requirement already satisfied: packaging>=20.0 in /srv/conda/lib/python3.11/
site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (24.0)
Requirement already satisfied: pillow>=6.2.0 in /srv/conda/lib/python3.11/si
te-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (10.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in /srv/conda/lib/python3.1
1/site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (3.1.2)
Requirement already satisfied: python-dateutil>=2.7 in /srv/conda/lib/python
3.11/site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (2.9.0)
Requirement already satisfied: pytz>=2020.1 in /srv/conda/lib/python3.11/sit
e-packages (from pandas>=1.2->seaborn) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in /srv/conda/lib/python3.11/s
ite-packages (from pandas>=1.2->seaborn) (2024.1)
Requirement already satisfied: six>=1.5 in /srv/conda/lib/python3.11/site-pa
ckages (from python-dateutil>=2.7->matplotlib!=3.6.1,>=3.4->seaborn) (1.16.
0)
Note: you may need to restart the kernel to use updated packages.
```

In [2]:
```
d = pd.read_csv('NY-House-Dataset.csv')
dd = pd.read_csv('income_NY.csv')
ddd=pd.read_csv('table-data.csv')
```

Comment: give your variables meaningful names

Cleaning and Filtering Data

I removed missing values to ensures the dataset is complete and reduces biases that could arise from incomplete data. I also filtered specific property types by focusing on condos, houses, townhouses, and multi-family homes, you create a niche dataset representing typical family or individual purchases, providing more relevant insights for potential buyers. Also, limited bedrooms to a maximum of 5 to makes the dataset more manageable and focuses on common residential properties, avoiding outliers like large mansions that could skew the analysis

Comment: What do you mean by "niche"?

In [3]:
```
# i want to know what are all the categories in the column name
ddd['name'].unique()
```

Out[3]:
```
array(['Albany County', 'Allegany County', 'Bronx County',
       'Broome County', 'Cattaraugus County', 'Cayuga County',
       'Chautauqua County', 'Chemung County', 'Chenango County',
       'Clinton County', 'Columbia County', 'Cortland County',
       'Delaware County', 'Dutchess County', 'Erie County',
       'Essex County', 'Franklin County', 'Fulton County',
       'Genesee County', 'Greene County', 'Hamilton County',
       'Herkimer County', 'Jefferson County', 'Kings County',
       'Lewis County', 'Livingston County', 'Madison County',
       'Monroe County', 'Montgomery County', 'Nassau County',
       'New York County', 'Niagara County', 'Oneida County',
       'Onondaga County', 'Ontario County', 'Orange County',
       'Orleans County', 'Oswego County', 'Otsego County',
       'Putnam County', 'Queens County', 'Rensselaer County',
       'Richmond County', 'Rockland County', 'St. Lawrence County',
       'Saratoga County', 'Schenectady County', 'Schoharie County',
       'Schuyler County', 'Seneca County', 'Steuben County',
       'Suffolk County', 'Sullivan County', 'Tioga County',
       'Tompkins County', 'Ulster County', 'Warren County',
       'Washington County', 'Wayne County', 'Westchester County',
       'Wyoming County', 'Yates County'], dtype=object)
```

In [4]:
```
# i only want certain counties to filter the data, because i only have
#this counties in one of my dataset and i want to match all datasets by cour
#i am renaming some columns to be easier to understand and match
#i want just two columns
#im removing missing values in the table
target_counties = ['New York County', 'Richmond County', 'Kings County', 'Qu
ddd_filtered = ddd[ddd['name'].isin(target_counties)]
ddd= ddd_filtered.rename(columns={'name': 'COUNTY', 'pop2024':'population'})
ddd= ddd[['population', 'COUNTY']]
ddd= ddd[~ddd.isna().any(axis=1)]
ddd
```

Out[4]:

	population	COUNTY
2	1331144	Bronx County
23	2532919	Kings County
30	1600359	New York County
40	2225834	Queens County
42	490016	Richmond County

In [5]:
```
#im removing missing values in the next dataset
# also want to know the categories in a column
dd= dd[~dd.isna().any(axis=1)]
dd['Table 1. Per Capita Personal Income, by County, 2020–2022'].unique()
```

Out[5]:
```
array(['United States', 'New York', 'Albany', 'Allegany', 'Bronx',
       'Broome', 'Cattaraugus', 'Cayuga', 'Chautauqua', 'Chemung',
       'Chenango', 'Clinton', 'Columbia', 'Cortland', 'Delaware',
       'Dutchess', 'Erie', 'Essex', 'Franklin', 'Fulton', 'Genesee',
       'Greene', 'Hamilton', 'Herkimer', 'Jefferson', 'Kings', 'Lewis',
       'Livingston', 'Madison', 'Monroe', 'Montgomery', 'Nassau',
       'Niagara', 'Oneida', 'Onondaga', 'Ontario', 'Orange', 'Orleans',
       'Oswego', 'Otsego', 'Putnam', 'Queens', 'Rensselaer', 'Richmond',
       'Rockland', 'St. Lawrence', 'Saratoga', 'Schenectady', 'Schoharie',
       'Schuyler', 'Seneca', 'Steuben', 'Suffolk', 'Sullivan', 'Tioga',
       'Tompkins', 'Ulster', 'Warren', 'Washington', 'Wayne',
       'Westchester', 'Wyoming', 'Yates'], dtype=object)
```

In [6]:
```
# basically doing the same as the other table
target_counties = ['New York County', 'Richmond County', 'Kings County', 'Qu
corrections = {
    'New York': 'New York County',
    'Bronx': 'Bronx County',
    'Kings': 'Kings County',
    'Queens': 'Queens County',
    'Richmond': 'Richmond County'
}
dd_filtered = dd[dd['Table 1. Per Capita Personal Income, by County, 2020-20

# Replace incorrect names with correct county names
dd_filtered['County'] = dd_filtered['Table 1. Per Capita Personal Income, by
# Removing NY State average
dd_filtered = dd_filtered.iloc[1:]
df1= dd_filtered.rename(columns={'Unnamed: 3': 'AVG_income_22', 'County': 'C
df1= df1[['AVG_income_22', 'COUNTY']]
df1
```

```
/tmp/ipykernel_405/2306163625.py:13: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  dd_filtered['County'] = dd_filtered['Table 1. Per Capita Personal Income,
by County, 2020-2022'].replace(corrections)
```

Out[6]:

	AVG_income_22	COUNTY
12	40,691	Bronx County
33	60,153	Kings County
40	186,848	New York County
50	54,261	Queens County
52	62,469	Richmond County

Comment: Is it possible that dropping rows with missing values is changing the sample you're working with? Could this matter?

In [7]:
```
# taking the features i want of the listing and removing any missing values
data= d[['TYPE','PRICE','BEDS', 'BATH', 'PROPERTYSQFT', 'SUBLOCALITY' ]]
data[~data.isna().any(axis=1)]
data
```

Out[7]:

	TYPE	PRICE	BEDS	BATH	PROPERTYSQFT	SUBLOCALITY
0	Condo for sale	315000	2	2.000000	1400.000000	Manhattan
1	Condo for sale	195000000	7	10.000000	17545.000000	New York County
2	House for sale	260000	4	2.000000	2015.000000	Richmond County
3	Condo for sale	69000	3	1.000000	445.000000	New York County
4	Townhouse for sale	55000000	7	2.373861	14175.000000	New York County
...	...	...	...	...	...	...
4796	Co-op for sale	599000	1	1.000000	2184.207862	New York
4797	Co-op for sale	245000	1	1.000000	2184.207862	Queens County
4798	Co-op for sale	1275000	1	1.000000	2184.207862	New York County
4799	Condo for sale	598125	2	1.000000	655.000000	Queens
4800	Co-op for sale	349000	1	1.000000	750.000000	Brooklyn

4801 rows × 6 columns

In [8]:
```
data['TYPE'].unique()
```

Out[8]:
```
array(['Condo for sale', 'House for sale', 'Townhouse for sale',
       'Co-op for sale', 'Multi-family home for sale', 'For sale',
       'Contingent', 'Land for sale', 'Foreclosure', 'Pending',
       'Coming Soon', 'Mobile house for sale', 'Condop for sale'],
      dtype=object)
```

In [9]:
```
desired_types = ['Condo for sale', 'House for sale', 'Townhouse for sale', '
df = data[data['TYPE'].isin(desired_types)]
df = df[df['BEDS'] <= 5]
# im filtering out to less than 5 beds because there where plenty of houses
#want that on my dataset because i want something more realistic
corrections = {
    'Manhattan': 'New York County',
    'New York': 'New York County',
    'East Bronx': 'Bronx County',
```

```
    'Brooklyn': 'Kings County',
    'The Bronx': 'Bronx County',
    'Queens': 'Queens County',
    'Staten Island': 'Richmond County',
    'Coney Island': 'Kings County',
    'Brooklyn Heights': 'Kings County',
    'Jackson Heights': 'Queens County',
    'Riverdale': 'Bronx County',
    'Rego Park': 'Queens County',
    'Fort Hamilton': 'Kings County',
    'Flushing': 'Queens County',
    'Dumbo': 'Kings County',
    'Snyder Avenue': 'Kings County'
}
df['SUBLOCALITY'] = df['SUBLOCALITY'].replace(corrections)
#i had to replace all sub localities with their accurate county

df= df.rename(columns={'SUBLOCALITY': 'COUNTY'})
df
```

Out[9]:

	TYPE	PRICE	BEDS	BATH	PROPERTYSQFT	COUNTY
0	Condo for sale	315000	2	2.000000	1400.000000	New York County
2	House for sale	260000	4	2.000000	2015.000000	Richmond County
3	Condo for sale	69000	3	1.000000	445.000000	New York County
5	House for sale	690000	5	2.000000	4004.000000	Kings County
6	Condo for sale	899500	2	2.000000	2184.207862	New York County
...	...	...	...	...	...	...
4787	Condo for sale	499000	3	1.000000	472.000000	New York County
4790	Condo for sale	789000	3	2.373861	800.000000	New York County
4792	Multi-family home for sale	1700000	3	7.000000	7854.000000	Kings County
4794	Condo for sale	945000	2	2.000000	903.000000	New York County
4799	Condo for sale	598125	2	1.000000	655.000000	Queens County

2378 rows × 6 columns

In [10]:
```
df['COUNTY'].unique()
```

Out[10]:
```
array(['New York County', 'Richmond County', 'Kings County',
       'Queens County', 'Bronx County'], dtype=object)
```

In [11]:
```
#now im merging all tables
df = pd.merge(df, df1, on='COUNTY', how='left')
df = pd.merge(df, ddd, on='COUNTY', how='left')
df
```

Out[11]:

	TYPE	PRICE	BEDS	BATH	PROPERTYSQFT	COUNTY	AVG_income_22
0	Condo for sale	315000	2	2.000000	1400.000000	New York County	186,848
1	House for sale	260000	4	2.000000	2015.000000	Richmond County	62,469
2	Condo for sale	69000	3	1.000000	445.000000	New York County	186,848
3	House for sale	690000	5	2.000000	4004.000000	Kings County	60,153
4	Condo for sale	899500	2	2.000000	2184.207862	New York County	186,848
...	...	...	...	...	...	...	...
2373	Condo for sale	499000	3	1.000000	472.000000	New York County	186,848
2374	Condo for sale	789000	3	2.373861	800.000000	New York County	186,848
2375	Multi-family home for sale	1700000	3	7.000000	7854.000000	Kings County	60,153
2376	Condo for sale	945000	2	2.000000	903.000000	New York County	186,848
2377	Condo for sale	598125	2	1.000000	655.000000	Queens County	54,261

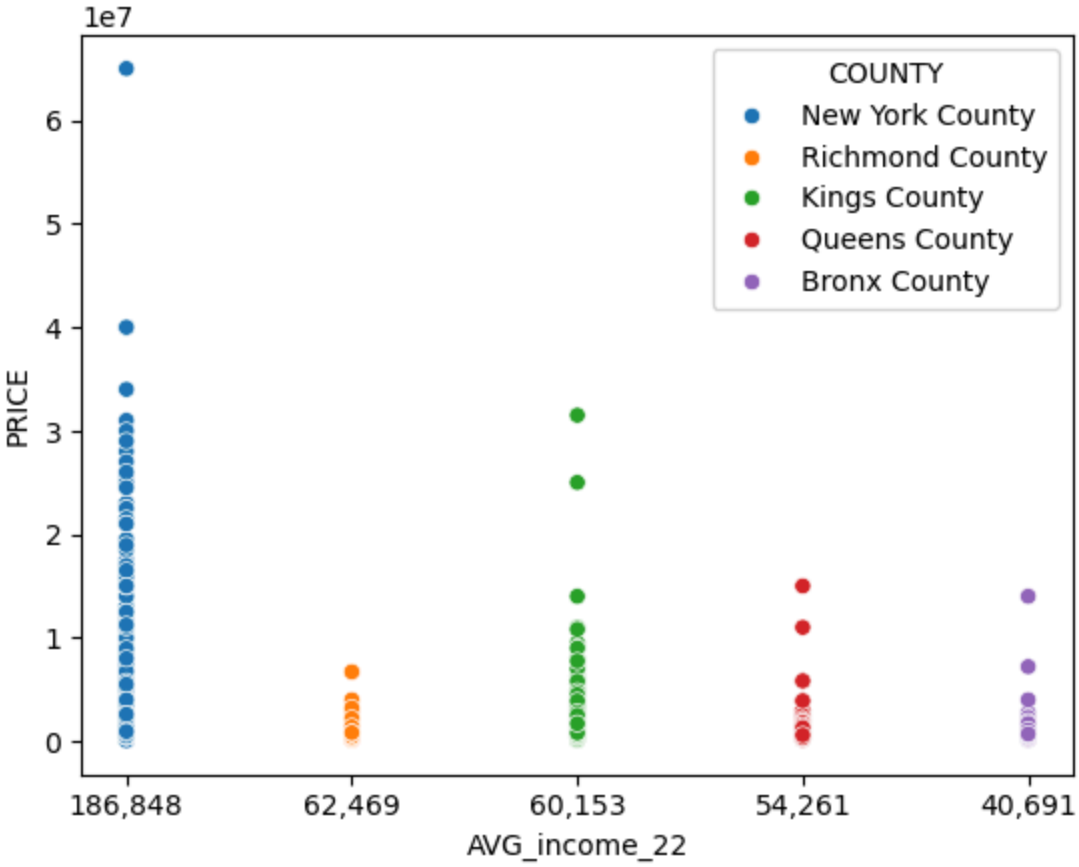
2378 rows × 8 columns

Visual Analysis

Plotting Prices Against Average Income with counties as different colots:

my purpose is to visualize the initial relationship between income and property prices to identify any apparent trends.

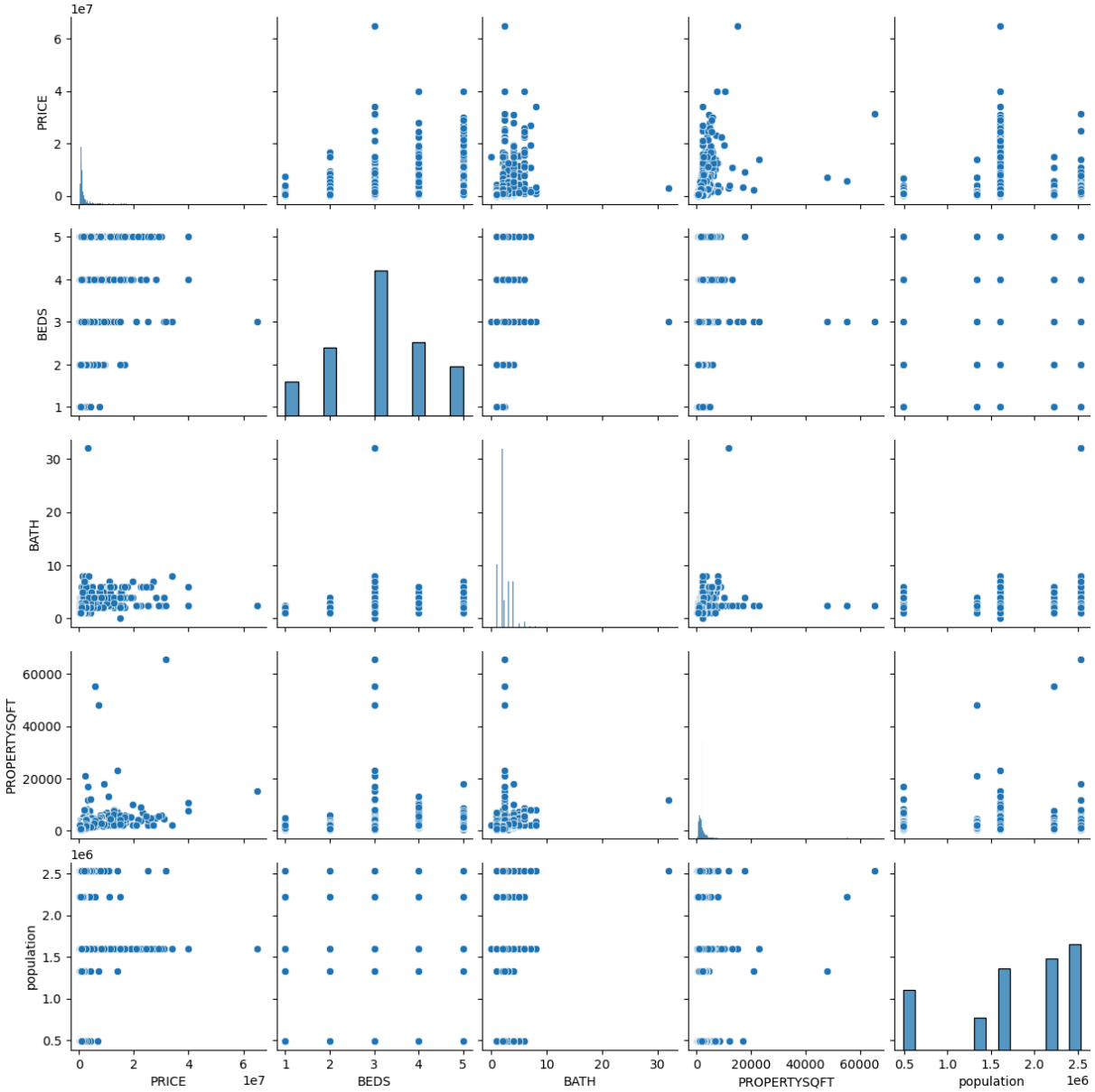
In [12]:
```
sns.scatterplot(x='AVG_income_22', y='PRICE', data=df, hue='COUNTY')
plt.show()
```



Comment: I see what you're getting at here, but a scatterplot is not a particularly good choice when there are only a few values for one of the variables

In [13]:
```
sns.pairplot(df)
```

Out[13]:
```
<seaborn.axisgrid.PairGrid at 0x7abe4d772e10>
```



New York County stands out with the highest prices and incomes, suggesting a potential correlation in this county.

Comment: You only have one observation for this county, how could there be a correlation in the county?

In [14]:
```
## i want to turn the average income column into integers, as i failed to tr
# to standard units because it was as strings and i had to replace the comma
df1 = df.copy()
df1['AVG_income_22'] = df1['AVG_income_22'].str.replace(',', '').astype(int)
df1
```

Out[14]:

	TYPE	PRICE	BEDS	BATH	PROPERTYSQFT	COUNTY	AVG_income_22
0	Condo for sale	315000	2	2.000000	1400.000000	New York County	186848
1	House for sale	260000	4	2.000000	2015.000000	Richmond County	62469
2	Condo for sale	69000	3	1.000000	445.000000	New York County	186848
3	House for sale	690000	5	2.000000	4004.000000	Kings County	60153
4	Condo for sale	899500	2	2.000000	2184.207862	New York County	186848
...	...	...	...	...	...	...	...
2373	Condo for sale	499000	3	1.000000	472.000000	New York County	186848
2374	Condo for sale	789000	3	2.373861	800.000000	New York County	186848
2375	Multi-family home for sale	1700000	3	7.000000	7854.000000	Kings County	60153
2376	Condo for sale	945000	2	2.000000	903.000000	New York County	186848
2377	Condo for sale	598125	2	1.000000	655.000000	Queens County	54261

2378 rows × 8 columns

Converting to Standard Units

im now converting numerical columns into standard units (z-scores) to ensure that all features are on the same scale. This is important for my analysis as it prevents variables with larger scales from disproportionately and affecting my model

In [15]:
```
# I will now proceed to look for a linear regression, and to do so i will co
#columns to Standard units to have more accurate results
# when i tried to plot the results i realized that the colunm income was str
```

```
# i also had to remove the commas to do so
scaler = StandardScaler()
df1['PRICE SU'] = scaler.fit_transform(df1[['PRICE']])
df1['AVG_income_22 SU'] = scaler.fit_transform(df1[['AVG_income_22']])
df1['PROPERTYSQFT SU'] = scaler.fit_transform(df1[['PROPERTYSQFT']])
df1['population SU'] = scaler.fit_transform(df1[['population']])
df1['BEDS SU'] = scaler.fit_transform(df1[['BEDS']])
df1['BATH SU'] = scaler.fit_transform(df1[['BATH']])
df1 = df1.drop(columns=['PRICE', 'AVG_income_22', 'PROPERTYSQFT', 'populatic
#df= df[features]
df1
```

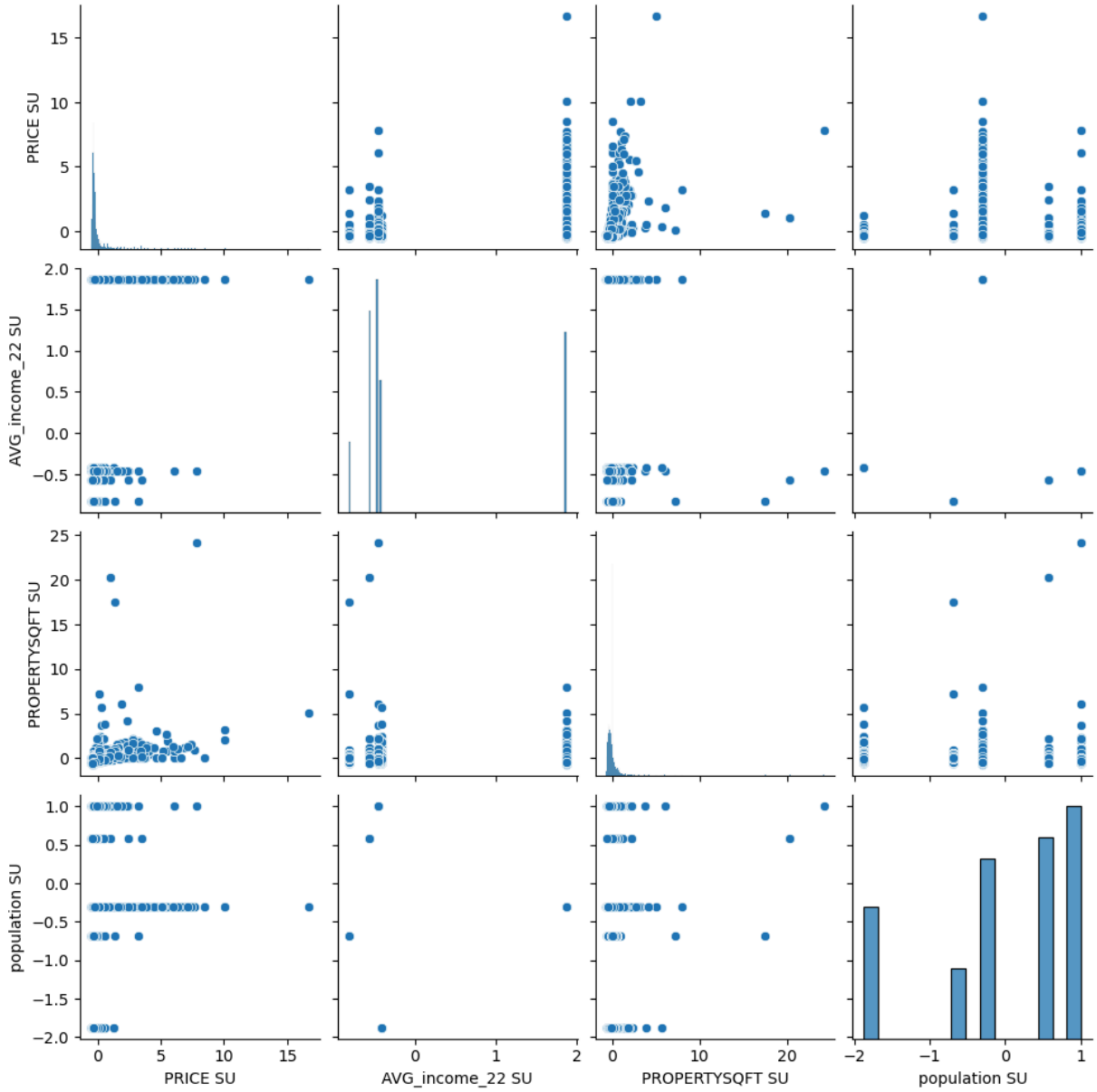
Out[15]:

	TYPE	COUNTY	PRICE SU	AVG_income_22 SU	PROPERTYSQFT SU	population SU	BI
0	Condo for sale	New York County	-0.444532	1.868382	-0.260074	-0.306741	-0.
1	House for sale	Richmond County	-0.459114	-0.421550	-0.025429	-1.873505	0.
2	Condo for sale	New York County	-0.509754	1.868382	-0.624443	-0.306741	-0.
3	House for sale	Kings County	-0.345107	-0.464190	0.733449	1.009161	1.
4	Condo for sale	New York County	-0.289562	1.868382	0.039130	-0.306741	-0.
...	...	...	...	...	...	...	
2373	Condo for sale	New York County	-0.395747	1.868382	-0.614141	-0.306741	-0.
2374	Condo for sale	New York County	-0.318859	1.868382	-0.488997	-0.306741	-0.
2375	Multi-family home for sale	Kings County	-0.077323	-0.464190	2.202368	1.009161	-0.
2376	Condo for sale	New York County	-0.277498	1.868382	-0.449698	-0.306741	-0.
2377	Condo for sale	Queens County	-0.369466	-0.572667	-0.544320	0.575844	-0.

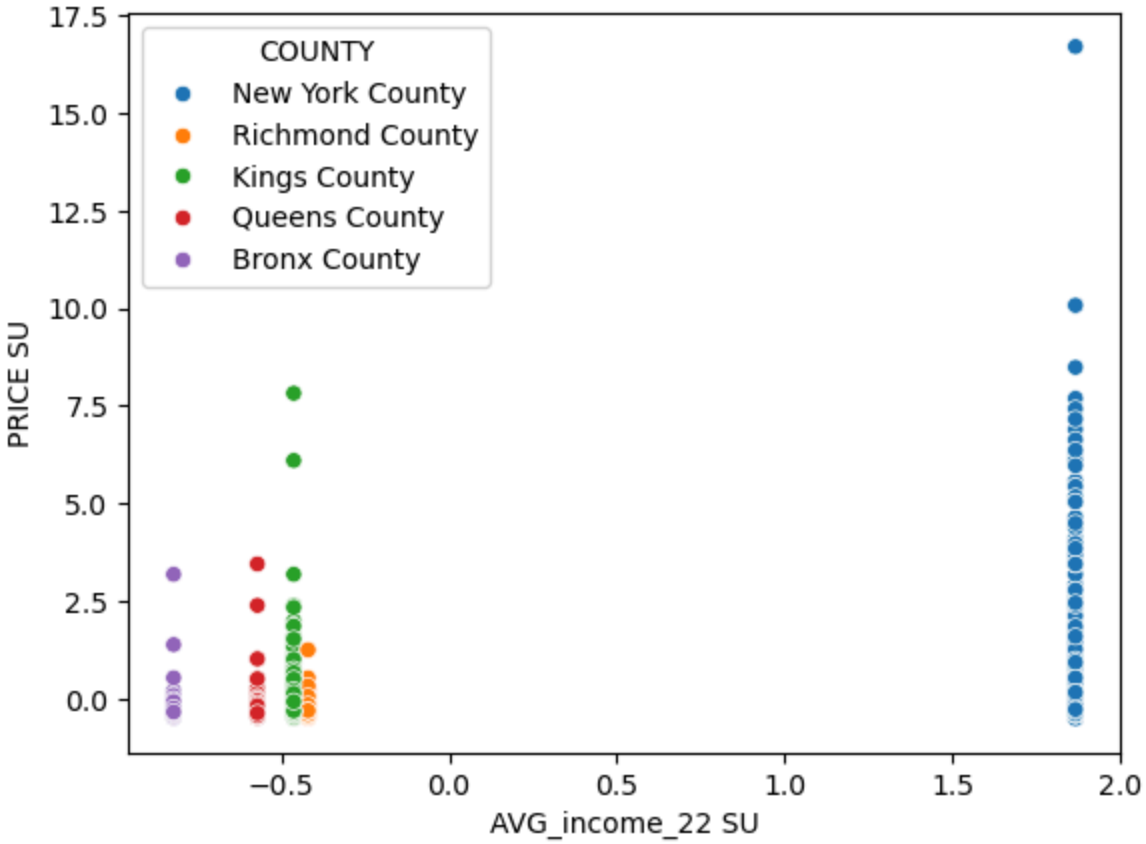
2378 rows × 8 columns

In [16]:
```
sns.pairplot(df1[['PRICE SU','AVG_income_22 SU','PROPERTYSQFT SU','populatic
```

Out[16]:
```
<seaborn.axisgrid.PairGrid at 0x7abe8f175210>
```



In [17]:
```
sns.scatterplot(x='AVG_income_22 SU', y='PRICE SU', data=df1, hue='COUNTY')
plt.show()
```



In [18]:
```
df1.head()
```

Out[18]:

	TYPE	COUNTY	PRICE SU	AVG_income_22 SU	PROPERTYSQFT SU	population SU	BEDS
0	Condo for sale	New York County	-0.444532	1.868382	-0.260074	-0.306741	-0.973
1	House for sale	Richmond County	-0.459114	-0.421550	-0.025429	-1.873505	0.801
2	Condo for sale	New York County	-0.509754	1.868382	-0.624443	-0.306741	-0.085
3	House for sale	Kings County	-0.345107	-0.464190	0.733449	1.009161	1.689
4	Condo for sale	New York County	-0.289562	1.868382	0.039130	-0.306741	-0.973

im converting my categorical columns to dummy variables to enable the inclusion of categorical data (e.g., county, property type) in the regression model by converting them into a numerical format

In [19]:
```
df1 = pd.get_dummies(df1, columns=['TYPE', 'COUNTY'], drop_first=True)
df1
```

Out[19]:

	PRICE SU	AVG_income_22 SU	PROPERTYSQFT SU	population SU	BEDS SU	BATH SU
0	-0.444532	1.868382	-0.260074	-0.306741	-0.973553	-0.287445
1	-0.459114	-0.421550	-0.025429	-1.873505	0.801838	-0.287445
2	-0.509754	1.868382	-0.624443	-0.306741	-0.085858	-1.100532
3	-0.345107	-0.464190	0.733449	1.009161	1.689533	-0.287445
4	-0.289562	1.868382	0.039130	-0.306741	-0.973553	-0.287445
...	...	...	...	...	...	...
2373	-0.395747	1.868382	-0.614141	-0.306741	-0.085858	-1.100532
2374	-0.318859	1.868382	-0.488997	-0.306741	-0.085858	0.016536
2375	-0.077323	-0.464190	2.202368	1.009161	-0.085858	3.777988
2376	-0.277498	1.868382	-0.449698	-0.306741	-0.973553	-0.287445
2377	-0.369466	-0.572667	-0.544320	0.575844	-0.973553	-1.100532

2378 rows × 13 columns

Initial Linear Regression Model - Training and Testing the Model:

To begin, I selected the relevant features and target variable for my linear regression model. The features included average income in standardized units and dummy variables representing the counties (Kings County, New York County, Queens County, and Richmond County). The target variable was the property price, also in standardized units. Using these, I split the dataset into training and testing sets, with 70% of the data allocated for training and 30% for testing.

Next, I initialized the linear regression model and trained it using the training data. This involved fitting the model to the training features and target variable. After training, I used the model to make predictions on the testing set.

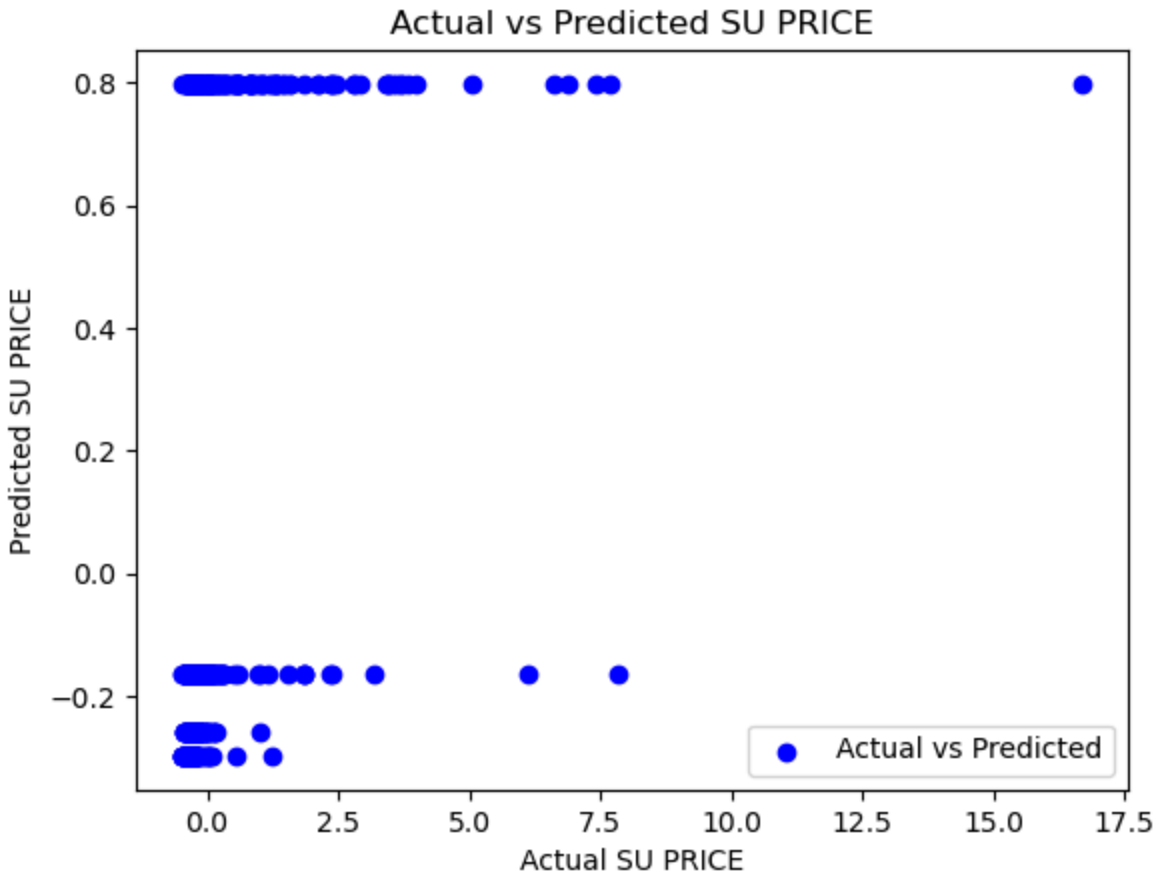
In [20]:
```
# now I will run a linear regression model by training 70% of my data and us
# I will print r^2 to see how accurate is my data

x = df1[['AVG_income_22 SU', 'COUNTY_Kings County', 'COUNTY_New York County'
y = df1['PRICE SU']
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, rar
model = LinearRegression()
model.fit(x_train, y_train)
```

```
y_pred = model.predict(x_test)

# Evaluate the model
r2 = r2_score(y_test, y_pred)


print(f'R-squared: {r2}')
```

```
R-squared: 0.14535471215990858
```

To evaluate the model's performance, I calculated the R-squared value. The resulting R-squared value was 0.145, indicating that the model explains only 14.7% of the variance in property prices based on average income and county. This low R-squared value suggests that the model has weak predictive power, meaning that other significant factors influencing property prices were not captured by this model.

In [21]:
```
plt.scatter(y_test, y_pred, color='blue', label='Actual vs Predicted')
plt.xlabel('Actual SU PRICE')
plt.ylabel('Predicted SU PRICE')
plt.legend()
plt.title('Actual vs Predicted SU PRICE')
plt.show()
```



To evaluate the models performance, I plotted the actual versus predicted property prices on standard units scale using a scatter plot. This visualization helps to see how closely the predictions match the actual values.

second su linear Model

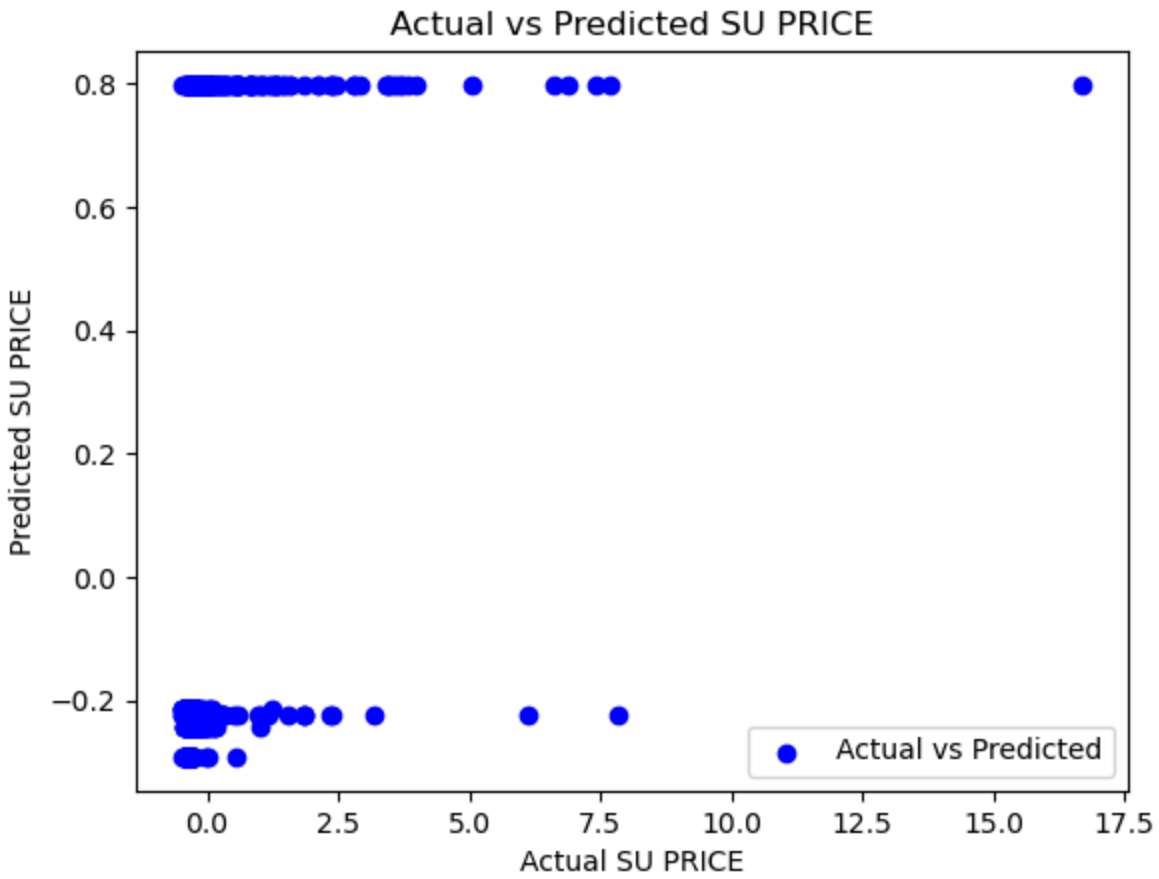
my purpose this time was to isolate the impact of New York County, given its high prices and incomes, to see if a more focused dataset improves the model's accuracy. I used the same way for calculating

In [22]:
```
x = df1[['AVG_income_22 SU', 'COUNTY_New York County' ]]
y = df1['PRICE SU']
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, rar
model = LinearRegression()
model.fit(x_train, y_train)
y_pred = model.predict(x_test)

# Evaluate the model
r2 = r2_score(y_test, y_pred)


print(f'R-squared: {r2}')
```

```
R-squared: 0.1398915463787599
```

The R-squared value slightly decreases to 0.139, indicating that even within a single county, average income alone is not a strong predictor of property prices.

In [23]:
```
plt.scatter(y_test, y_pred, color='blue', label='Actual vs Predicted')
plt.xlabel('Actual SU PRICE')
plt.ylabel('Predicted SU PRICE')
plt.legend()
plt.title('Actual vs Predicted SU PRICE')
plt.show()
```



Third model Incorporating Additional Features

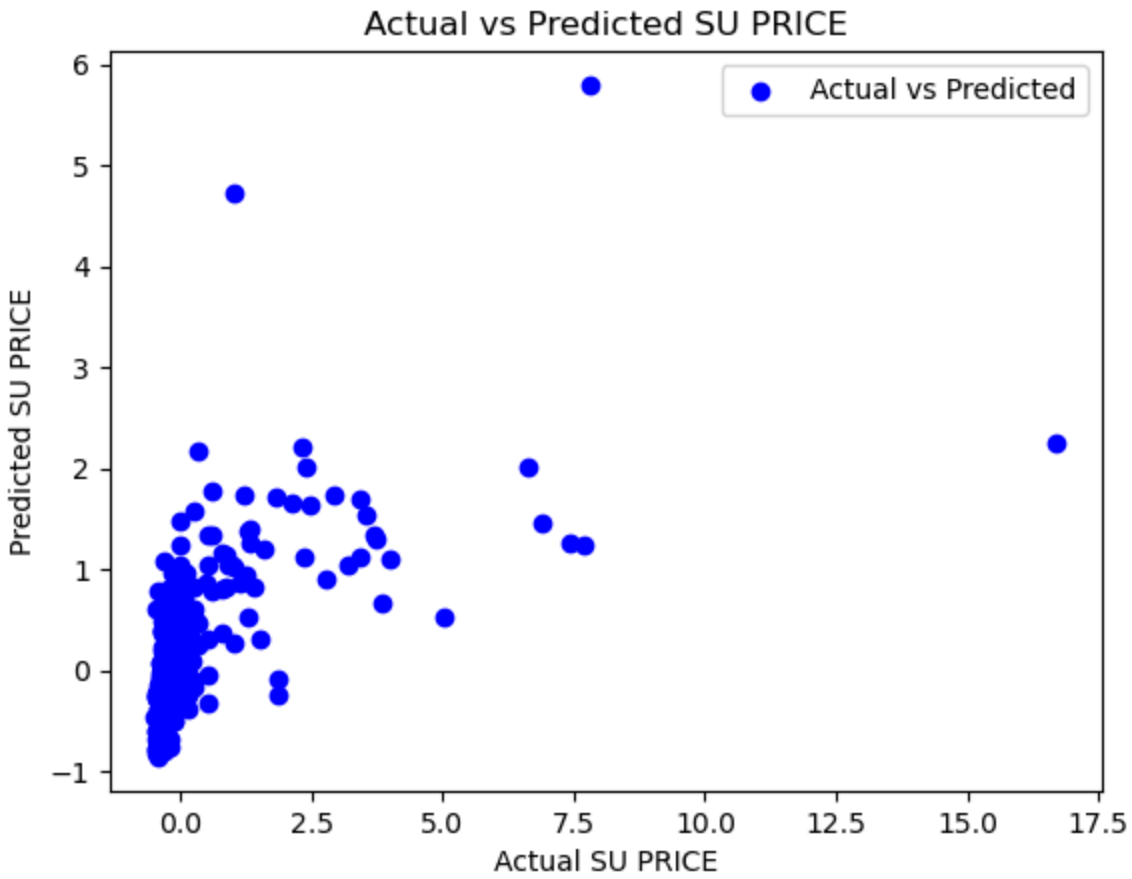
This time my purpose is to add the Dummy Variables for County and Property Type to examine if including more categorical features enhances the model's predictive power by capturing more variability in the data.

In [24]:
```
# Define features and target variable
X = df1[['BEDS SU','BATH SU','PROPERTYSQFT SU', 'population SU', 'AVG_income
y = df1['PRICE SU']

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rar

# Create the model
model = LinearRegression()

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

# Make predictions
y_pred = model.predict(X_test)

r2 = r2_score(y_test, y_pred)


print(f'R-squared: {r2}')
```

```
R-squared: 0.3669759311618004
```

The R-squared value increases to 0.367, showing a moderate positive correlation and an improvement in the model's explanatory power.

In [25]:
```
plt.scatter(y_test, y_pred, color='blue', label='Actual vs Predicted')
plt.xlabel('Actual SU PRICE')
plt.ylabel('Predicted SU PRICE')
plt.legend()
plt.title('Actual vs Predicted SU PRICE')
plt.show()
```



Log Transformation and Standardization

Now i want to address potential non-linear relationships by log-transforming the features, which can often linearize relationships and improve model performance, then I standardize the values to ensure all features remain on the same scale post-transformation.

In [26]:
```
df.head()
```

Out[26]:

	TYPE	PRICE	BEDS	BATH	PROPERTYSQFT	COUNTY	AVG_income_22	popula
0	Condo for sale	315000	2	2.0	1400.000000	New York County	186,848	1600
1	House for sale	260000	4	2.0	2015.000000	Richmond County	62,469	490
2	Condo for sale	69000	3	1.0	445.000000	New York County	186,848	1600
3	House for sale	690000	5	2.0	4004.000000	Kings County	60,153	2532
4	Condo for sale	899500	2	2.0	2184.207862	New York County	186,848	1600

In [27]:
```
## just as i converted to standard units, now i am converting first to log

df['AVG_income_22'] = df['AVG_income_22'].str.replace(',', '').astype(int)
df['log_PRICE'] = np.log(df['PRICE'])
df['log_PROPERTYSQFT'] = np.log(df['PROPERTYSQFT'])
df['log_income'] = np.log(df['AVG_income_22'])
df['log_population'] = np.log(df['population'])
df['log_BEDS'] = np.log(df['BEDS'])
df['log_BATH'] = np.log(df['BATH'])
df = df.drop(columns=['PRICE', 'AVG_income_22', 'PROPERTYSQFT', 'population'
## getting the dummy variables of the categorical values to make them numeri
df = pd.get_dummies(df, columns=['TYPE', 'COUNTY'], drop_first=True)
df.head()
```

```
/srv/conda/lib/python3.11/site-packages/pandas/core/arraylike.py:399: Runtim
eWarning: divide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

Out[27]:

	log_PRICE	log_PROPERTYSQFT	log_income	log_population	log_BEDS	log_BATH
0	12.660328	7.244228	12.138051	14.285739	0.693147	0.693147
1	12.468437	7.608374	11.042426	13.102193	1.386294	0.693147
2	11.141862	6.098074	12.138051	14.285739	1.098612	0.000000
3	13.444447	8.295049	11.004647	14.744883	1.609438	0.693147
4	13.709594	7.689009	12.138051	14.285739	0.693147	0.693147

In [28]:
```
## now im converting the logs to standard units with the same method
scaler = StandardScaler()
df['log_price_SU'] = scaler.fit_transform(df[['log_PRICE']])
df['log_income_SU'] = scaler.fit_transform(df[['log_income']])
```

```
df['log_sqft_SU'] = scaler.fit_transform(df[['log_PROPERTYSQFT']])
df['log_population_SU'] = scaler.fit_transform(df[['log_population']])

df = df.drop(columns=['log_PRICE', 'log_income', 'log_PROPERTYSQFT', 'log_pc
df
```

Out[28]:

	log_BEDS	log_BATH	TYPE_House for sale	TYPE_Multi-family home for sale	TYPE_Townhouse for sale	COUNTY_K Co
0	0.693147	0.693147	False	False	False	
1	1.386294	0.693147	True	False	False	
2	1.098612	0.000000	False	False	False	
3	1.609438	0.693147	True	False	False	
4	0.693147	0.693147	False	False	False	
...	...	...	...	...	...	
2373	1.098612	0.000000	False	False	False	
2374	1.098612	0.864518	False	False	False	
2375	1.098612	1.945910	False	True	False	
2376	0.693147	0.693147	False	False	False	
2377	0.693147	0.000000	False	False	False	

2378 rows × 13 columns

Testing first log model with counties and income to price

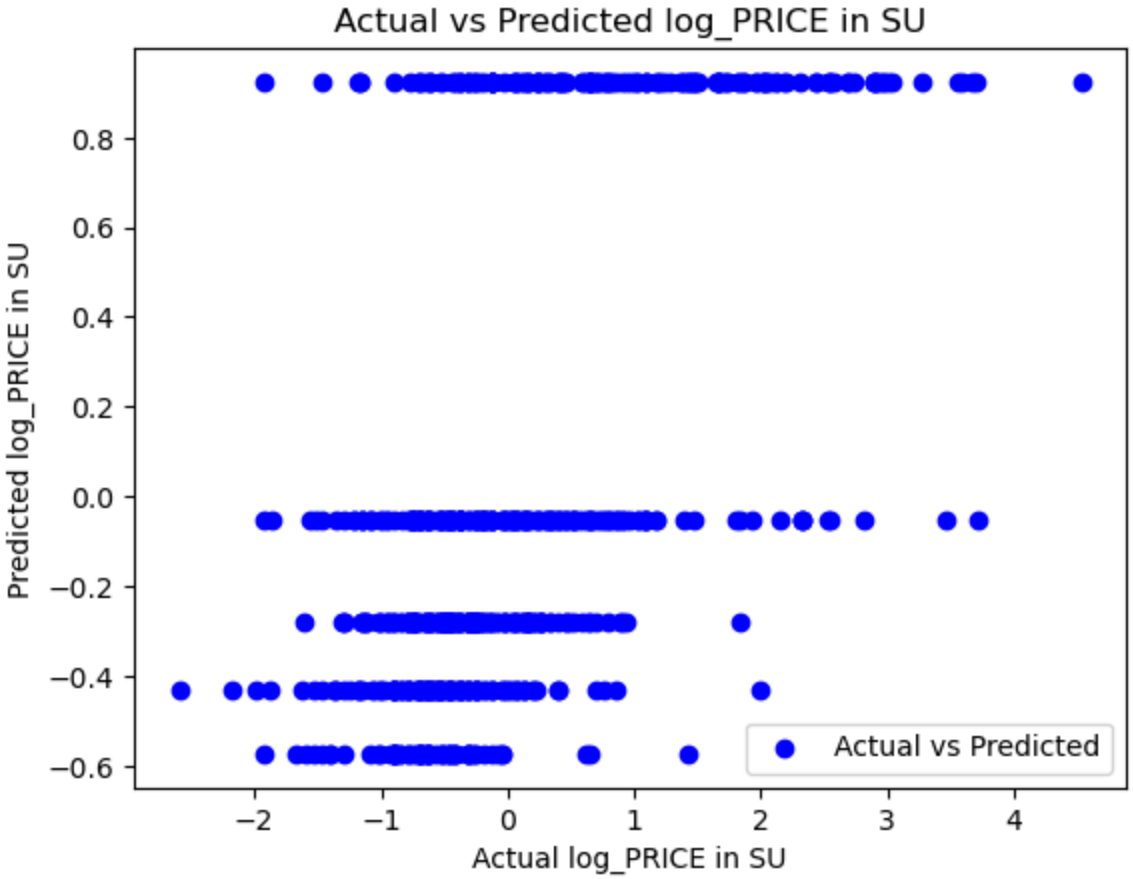
In [29]:
```
x = df[['log_income_SU', 'COUNTY_Kings County', 'COUNTY_New York County','CO
y = df['log_price_SU']
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, ran
model = LinearRegression()
model.fit(x_train, y_train)
y_pred = model.predict(x_test)

# Evaluate the model
r2 = r2_score(y_test, y_pred)


print(f'R-squared: {r2}')
```

```
R-squared: 0.30950312138500846
```

An R-squared of 0.309 indicates a moderate but still weak relationship.

In [30]:
```
plt.scatter(y_test, y_pred, color='blue', label='Actual vs Predicted')
plt.xlabel('Actual log_PRICE in SU')
```

```
plt.ylabel('Predicted log_PRICE in SU')
plt.legend()
plt.title('Actual vs Predicted log_PRICE in SU')
plt.show()
```



second log model with NY county and income to price

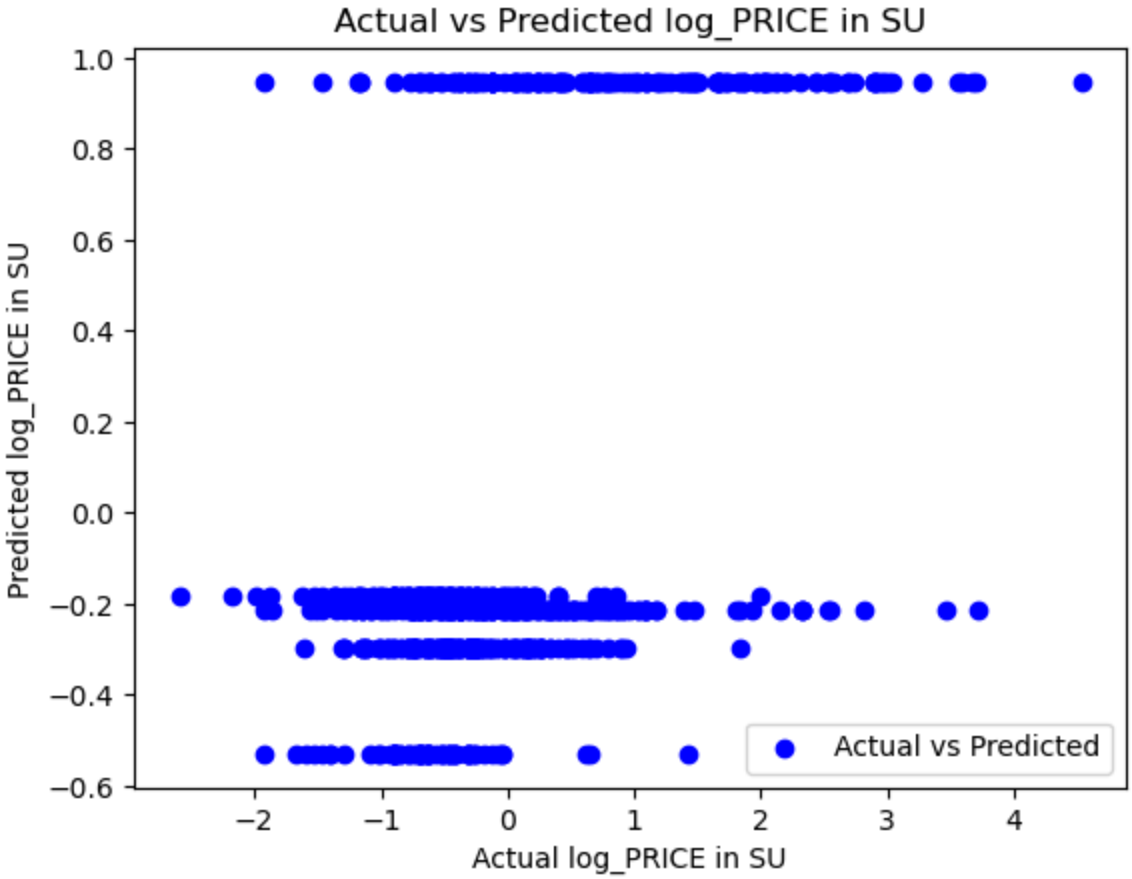
In [31]:
```
x = df[['log_income_SU', 'COUNTY_New York County' ]]
y = df['log_price_SU']
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, rar
model = LinearRegression()
model.fit(x_train, y_train)
y_pred = model.predict(x_test)

# Evaluate the model
r2 = r2_score(y_test, y_pred)


print(f'R-squared: {r2}')
```

```
R-squared: 0.27471641720804885
```

Again .275 indicates that even within a single county, average income alone is not a strong predictor of property prices.

In [32]:
```
plt.scatter(y_test, y_pred, color='blue', label='Actual vs Predicted')
plt.xlabel('Actual log_PRICE in SU')
```

```
plt.ylabel('Predicted log_PRICE in SU')
plt.legend()
plt.title('Actual vs Predicted log_PRICE in SU')
plt.show()
```



Third log model using income and sqft

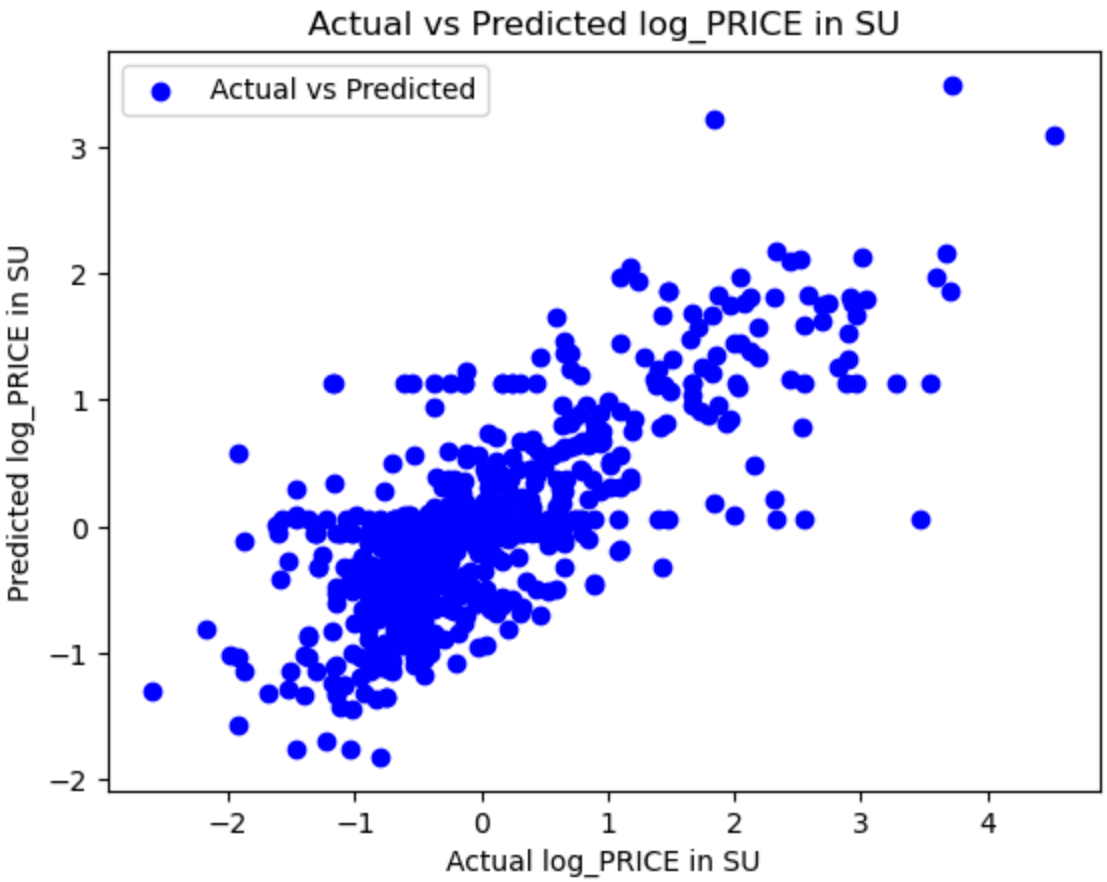
I decided to create a model using square footage and income to predict property prices because property size is a critical factor valuing properties and i wanted to see how much weight alone with income it had on the model.

In [33]:
```
x = df[['log_income_SU','log_sqft_SU' ]]
y = df['log_price_SU']
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, rar
model = LinearRegression()
model.fit(x_train, y_train)
y_pred = model.predict(x_test)

# Evaluate the model
r2 = r2_score(y_test, y_pred)


print(f'R-squared: {r2}')
```

```
R-squared: 0.572340970788844
```

The resulting R-squared value of 0.572 demonstrated a stronger moderate relationship compared to the initial model, highlighting that including square footage significantly improves the model's ability to predict prices and the income.

In [34]:
```
plt.scatter(y_test, y_pred, color='blue', label='Actual vs Predicted')
plt.xlabel('Actual log_PRICE in SU')
plt.ylabel('Predicted log_PRICE in SU')
plt.legend()
plt.title('Actual vs Predicted log_PRICE in SU')
plt.show()
```



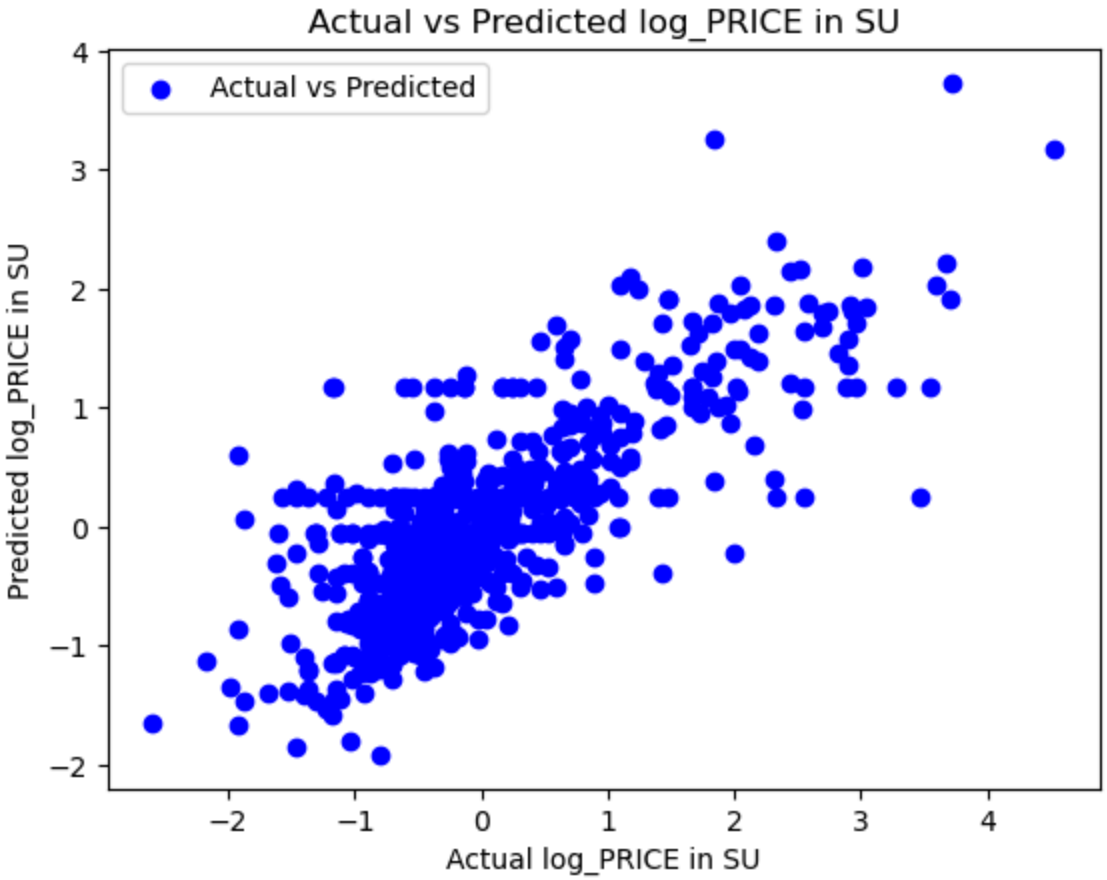
Final Model using income, sqft and counties

In [35]:
```
x = df[['log_income_SU','log_sqft_SU', 'COUNTY_Kings County', 'COUNTY_New Yo
y = df['log_price_SU']
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, ran
model = LinearRegression()
model.fit(x_train, y_train)
y_pred = model.predict(x_test)

# Evaluate the model
r2 = r2_score(y_test, y_pred)


print(f'R-squared: {r2}')
```

```
R-squared: 0.5963559290853894
```

An R-squared of 0.596 represents the best performance, showing a moderate and more accurate relationship between features and property prices.

In [36]:
```
plt.scatter(y_test, y_pred, color='blue', label='Actual vs Predicted')
plt.xlabel('Actual log_PRICE in SU')
plt.ylabel('Predicted log_PRICE in SU')
plt.legend()
plt.title('Actual vs Predicted log_PRICE in SU')
plt.show()
```



Final Summary

In conclusion, my analysis aimed to explore the relationship between average income and property prices in New York counties, incorporating various data sources and modeling techniques. Initially, the linear regression model using only income and county data showed a weak predictive power with an R-squared value of 0.145, suggesting other significant factors were influencing property prices. By refining the model to include additional features such as square footage and utilizing data transformations, I observed notable improvements. The model that included square footage and income yielded an R-squared of 0.572, highlighting the critical role of property size in price prediction. Further enhancement with dummy variables for county and property type achieved an R-squared of 0.596, demonstrating a more robust and accurate model.

Visualizations of the model's performance, both in logarithmic and original scales, provided clear insights into the predictive accuracy and areas for improvement. Overall, this analysis underscores the complexity of the real estate market.

Comment: This analysis shows a lot of depth and thought, especially in the production of the dataset. One weakness is that you only effectively have five values for the income variable because you're averaging over (large and diverse) counties.