Devin Lim

DATA 5322 Written Homework 2: Support Vector Machines

April 29, 2024

Comparing Predictive Factors of Homeownership in Single-Family Households Across Age Groups

Abstract

Homeownership rates in the U.S. have fallen since its peak in 2004[1], and some researchers have argued that the 'real' homeownership gap between young adults and past generations is much larger than traditional measure shows, causing a significant wealth gap compared to older generations who were able to build wealth through housing equity[2]. Using US Census data collected through IPUMS USA[3], this paper compares the changes in the predictive power of predictors for homeownership status across 3 age groups: Young Adults (aged 18-34), Middle-Aged Adults (aged 35-64), and Older Adults (aged 65+). These predictors are comprised of housing characteristics (such as age of structure), demographic characteristics (such as population density), and personal characteristics (such as educational attainment). By using Support Vector Classifiers with linear, radial, and polynomial kernels, results show that although there are commonalities between age groups, the relative importance of factors shift substantially. For instance, while personal income is the second most important factor for young adult homeownership, it shifts to becoming the second least important factor for older adult homeownership. This suggests that focused effort is necessary to assist young adults in closing the homeownership gap.

Introduction

Homeownership rates in the U.S. have dropped from a peak of 69.2 percent in 2004 to 65.7 percent in 2023[1]. By traditional measures, homeownership rates for young adults have declined from 45.0 percent in 1990 to 41.6 percent in 2021. However, traditional measures typically consider only one type of living arrangement: "living independently as a one-person household or as the household head with other individuals in a household." When we consider another type of living arrangement – being the spouse of a household head, this becomes a "15.4 percentage-point decline between 1990 and 2021". This decline is met with an increase in "in the share of young adults living with parents (from 12.5 percent to 20.2 percent) and those living with others (from 10.7 percent to 18.3 percent)"[2]. Other reports have examined the reasons behind falling homeownership rates among the younger generation, attributing it to the costs of higher education, house prices, and rents that have grown faster than incomes[4]. The focus of this paper is to not to analyze the factors behind falling homeownership rates directly, but rather to examine the differences in the power of predictive factors across age groups.

The dataset used in this analysis is a subset of the IPUMS USA 2023 census, containing 10 predictor variables: "Population-weighted density of PUMA", "Number of rooms", "Age of structure", "Number of bedrooms", "Number of families in household", "Number of couples in household", "Age", "Marital status", "Educational attainment", and "Total personal income". IPUMS USA "is a project dedicated to collecting and distributing United States census data" ran out of University of Minnesota and it contains data from 1790 to the present, available by request[3].

Various Support Vector Machine models were built against a binary response variable ("Owned'" or "Rented") using the same set of predictors but different segments of the data that were split by age groups. The models were then compared by its efficacy and interpretability. Background information on the data preparation process and machine learning methods along with discussion on the findings are provided below.

Background

Support Vector Machines

Support Vector Machines (SVMs) are supervised learning models whose most basic version is a Maximal Margin Classifier (MMC). An MMC aims to find a separating hyperplane (a line in 2D) that separates two classes such that the distance, known as the margin, between the line and its closest points is maximized. Support vectors are points that lie on the margin. An MMC do not allow for any points to be misclassified.

Because SVMs are based on distances, the “hyperplane is influenced by the scale of the input features and it’s therefore recommended that data be standardized … prior to SVM model training”[6].

Support Vector Classifiers

A Support Vector Classifier (SVC) is a more specific type of SVMs used for classification problems. It is like an MMC except that it allows for misclassified points. This lets the classifier to improve the classification of majority of the points. An SVC can be made from different types of kernels, but all forms of SVCs can be tuned by adjusting its “C”, which is either the “budget” for misclassified points or the “cost” of violation of the margin depending on the library or context. A good starting point for “C” is a very small value like 0.0001 then increment it exponentially/logarithmically like 0.001, 0.01, etc.

Kernels

A Kernel can be generally thought of as a function that maps the relationship between two points.

Linear Kernel

A Linear Kernel creates a linear hyperplane to separate classes. Suitable (only) for linearly separable data and computes reasonably fast. It can be tuned by adjusting its “C”. Higher levels of “C” (as in “cost”) leads to more complex models as it penalizes misclassifications further. For instance, Figure 1 is an SVC created with a linear kernel of

C=0.01 (a very low "cost"), hence it allows many red points which should be classified on the left side to be misclassified on the right.

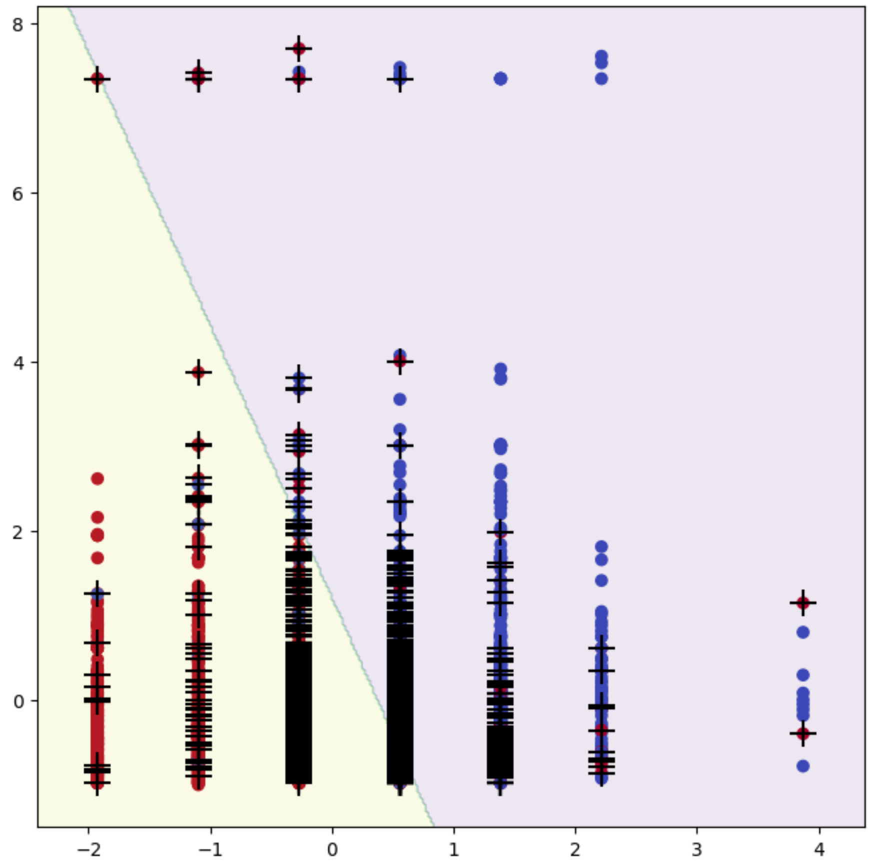


Figure 1. Linear SVC example using "Number of bedrooms" and "Total personal income"

Radial Kernel

A Radial Kernel creates a curvy or circular hyperplane to separate classes. Suitable for non-linear relationships, though it is more computationally expensive to fit. It can be tuned by adjusting its "C" and "gamma", which defines how far the influence of a single data point reaches. Higher levels of "gamma" limits the influence of a single data point which leads to more complex models.

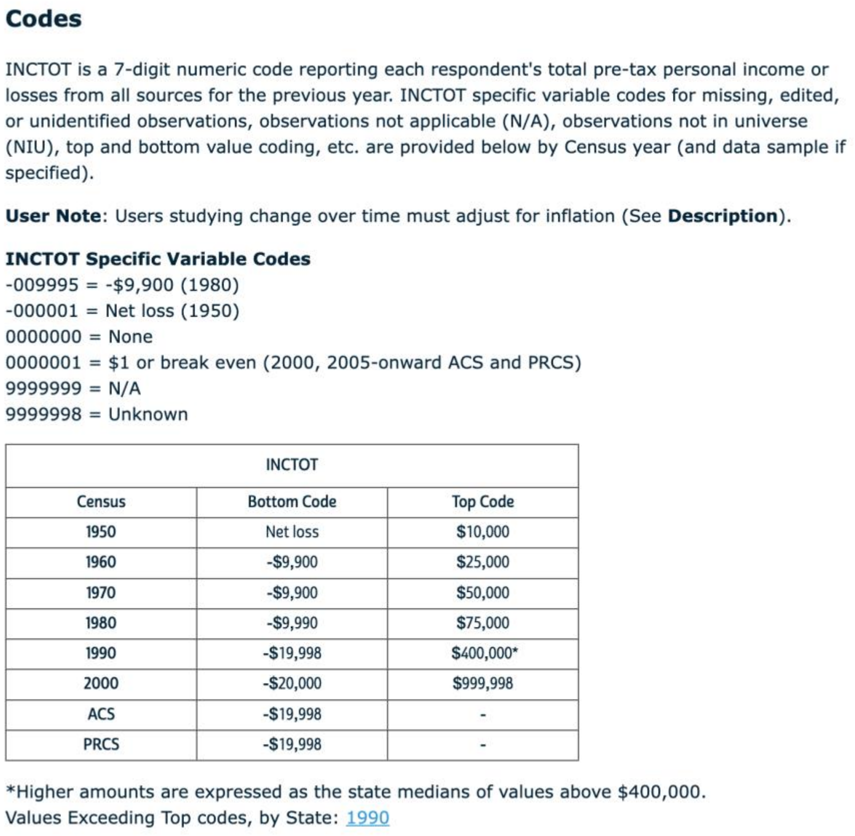
Polynomial Kernel

A Polynomial Kernel creates a polynomial hyperplane to separate classes. Suitable for more complex non-linear relationships as it can model increasingly complex interactions between features. It is even more computationally expensive to train than radial kernels, hence radial kernels are usually better general-purpose kernels. It can be tuned by adjusting its “C”, “degree”, and “coef0”. “degree” is the degree of polynomials, so a degree of 2 would result in a dot product of all data points raised to the power of two (lower values lead to simpler models). “coef0” adjusts the influence of higher-degree terms vs lower-degree terms (lower values weakens the higher-degree terms, leading to simpler models).

Methodology

Data Preparation

Each variable’s value in the dataset has been encoded, with their meanings searchable through IPUMS USA’s live search portal[7]. Some variables were numerical with certain values reserved for special meanings (Figure 2), while others were categorical.



Codes

INCTOT is a 7-digit numeric code reporting each respondent's total pre-tax personal income or losses from all sources for the previous year. INCTOT specific variable codes for missing, edited, or unidentified observations, observations not applicable (N/A), observations not in universe (NIU), top and bottom value coding, etc. are provided below by Census year (and data sample if specified).

User Note: Users studying change over time must adjust for inflation (See **Description**).

INCTOT Specific Variable Codes
-009995 = -$9,900 (1980)
-000001 = Net loss (1950)
0000000 = None
0000001 = $1 or break even (2000, 2005-onward ACS and PRCS)
9999999 = N/A
9999998 = Unknown

INCTOT		
Census	Bottom Code	Top Code
1950	Net loss	$10,000
1960	-$9,900	$25,000
1970	-$9,900	$50,000
1980	-$9,990	$75,000
1990	-$19,998	$400,000*
2000	-$20,000	$999,998
ACS	-$19,998	-
PRCS	-$19,998	-

*Higher amounts are expressed as the state medians of values above $400,000.
Values Exceeding Top codes, by State: 1990

Figure 2. IPUMS USA codebook example.[8]

The data preparation process starts with relabeling our columns to be human readable then subsetting our columns of interest and marking specific variable codes like 9999999 as missing data. Afterwards, the data is filtered to single family households whose dwelling is inhabited by 0 or 1 couples – this was done because 2 or 3 couples of the same family living in the same household are outliers. The missing data is then removed (11840 rows out of 68131, or 17.38 percent).

As the dataset contains information on dwellings that contain multiple people in each, they must be aggregated as having multiple data points on the same dwelling can skew the results. The aggregation method chosen was to aggregate by the head of household, defined as a person 18 or older with highest personal income.

Some variables were then consolidated into fewer categories. For example, "Educational Attainment" whose value range from 0 to 11 was grouped into five categories: "Less than high school", "High school", "Some college", "Bachelor's", and "Graduate". The same process was applied to "Marital status", which became: "Married", "Previously married", "Never married".

Finally, categorical variables were one-hot encoded and three separate dataframes were created by segmenting the original data per age group.

Modeling

For each age group, we trained Support Vector Classifiers (SVCs) with a linear, radial, and polynomial kernel. Prior to performing any modeling, the data was split 70-30 into a training and testing set and its features standardized. All models were trained with the same set of features and tuned with a 5-fold grid search cross validation. The parameters tuned are: "C" (applicable to all three kernels), "gamma" (applicable to radial kernels), and "degree" and "coef0" (applicable to polynomial kernels).

Feature importance analysis was done using linear kernel SVCs since both radial and polynomial kernels transforms data into higher dimensional space, complicating interpretation of predictors.

Two graphs are provided for each age group: one that includes each encoded feature because of one-hot encoding, color coded by the direction of the relationship, and another that includes sum-aggregated absolute values of the coefficients. The first is useful to compare and interpret predictors within an age group while the second is useful to interpret the predictors importance within and across age groups.

Results

Young Adults Home Ownership

SVM Kernel	Test Accuracy
Radial	79.76%
Polynomial	79.69%
Linear	79.20%

Table 1. Prediction accuracies for Young Adults Home Ownership by model.

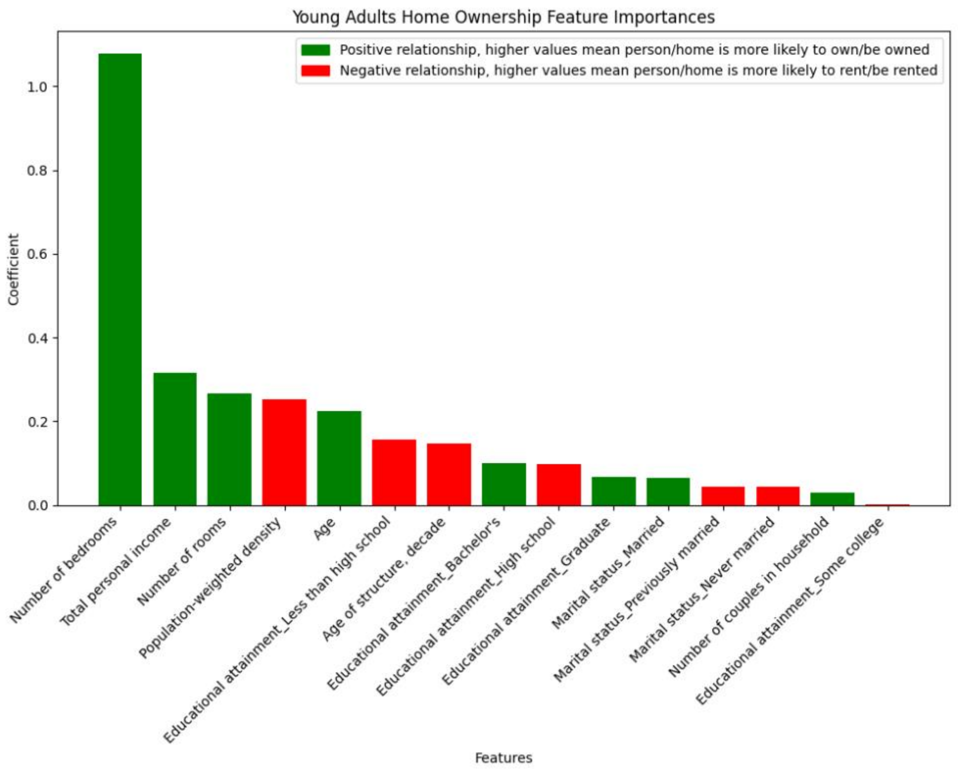


Figure 3. Correlation-coded feature importances for Young Adults Home Ownership.

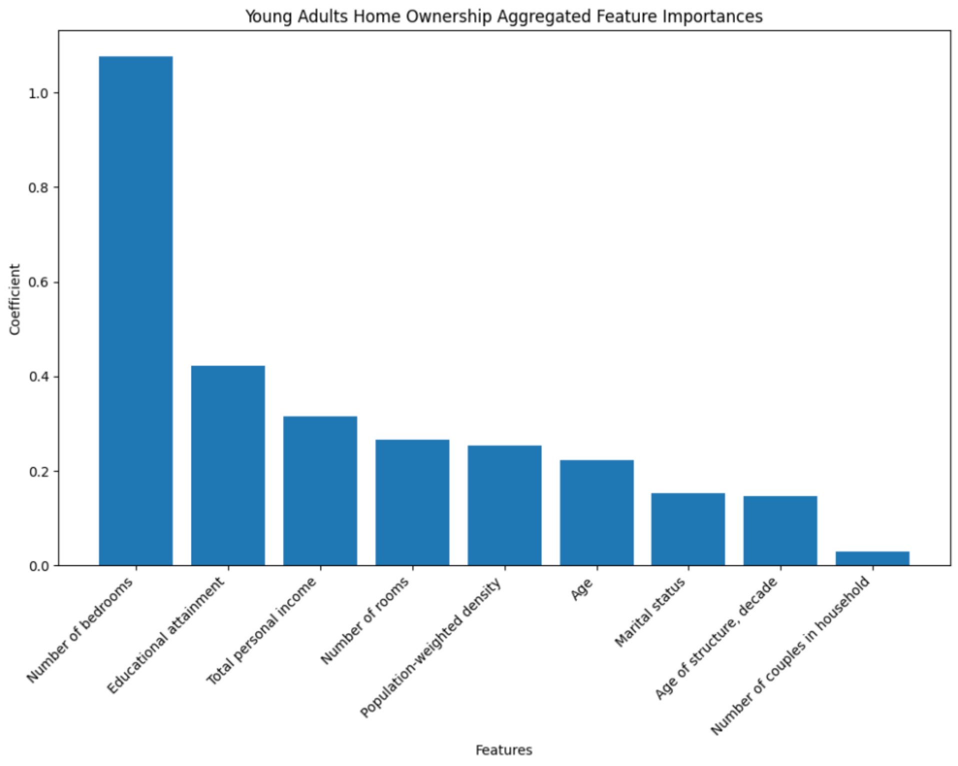


Figure 4. Aggregated feature importances for Young Adults Home Ownership.

Middle-Aged Adults Home Ownership

SVM Kernel	Test Accuracy
Radial	82.59%
Polynomial	82.87%
Linear	82.61%

Table 2. Prediction accuracies for Middle-Aged Adults Home Ownership by model.

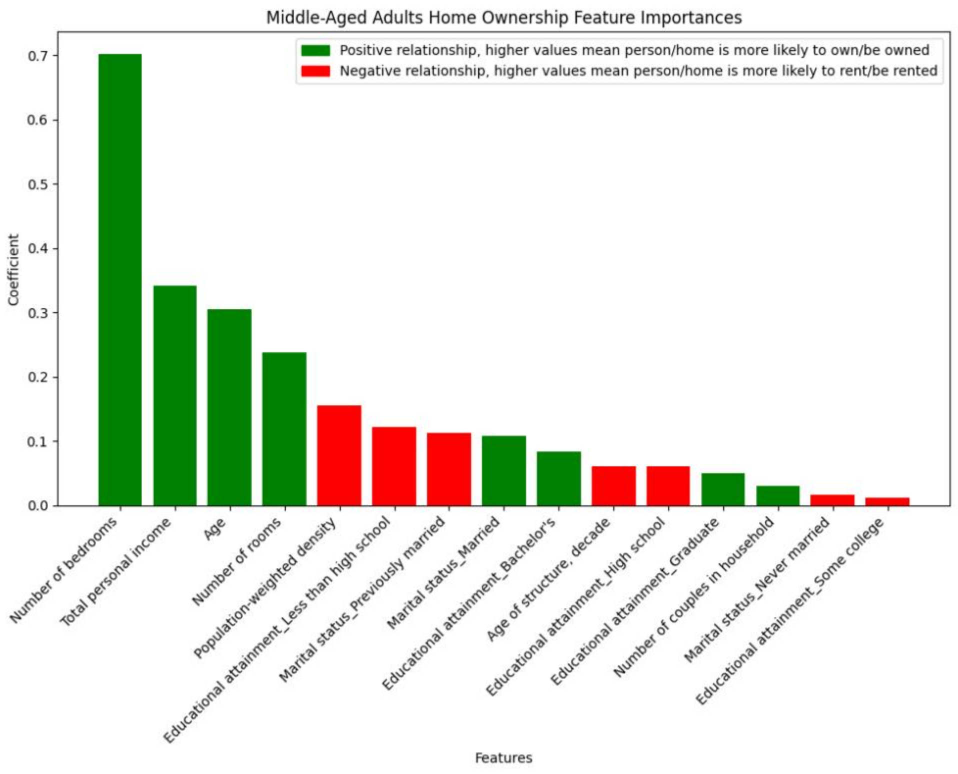


Figure 5. Correlation-coded feature importances for Middle-Aged Adults Home Ownership.

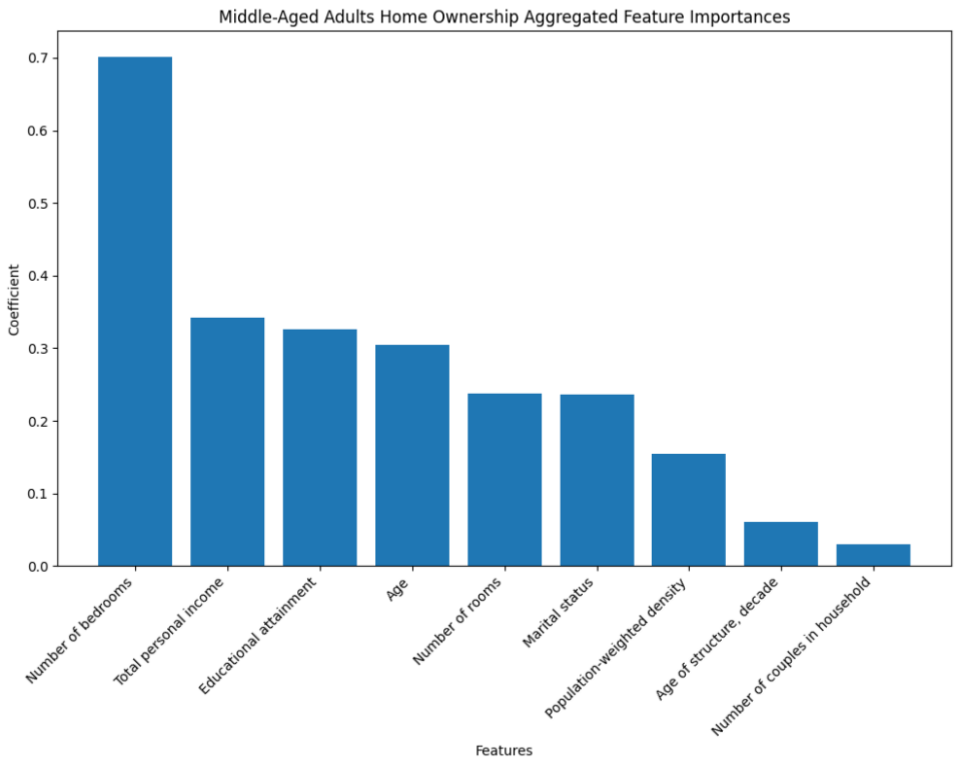


Figure 6. Aggregated feature importances for Middle-Aged Adults Home Ownership.

Older Adults Home Ownership

SVM Kernel	Test Accuracy
Radial	87.06%
Polynomial	87.06%
Linear	87.10%

Table 3. Prediction accuracies for Older Adults Home Ownership by model.

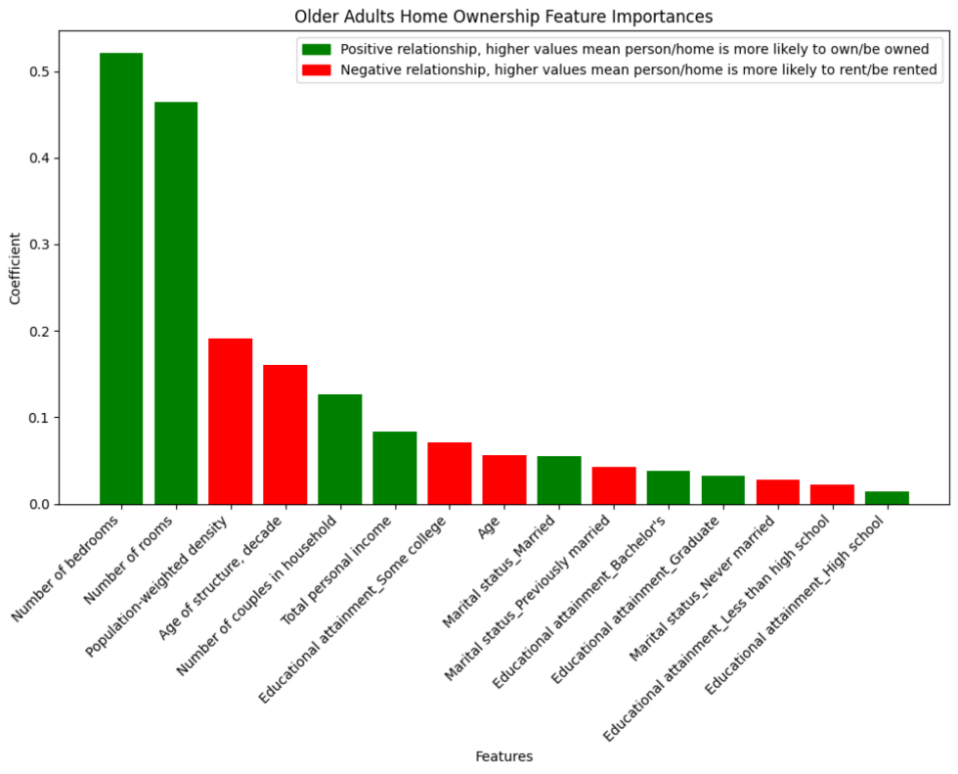


Figure 7. Correlation-coded feature importances for Older Adults Home Ownership.

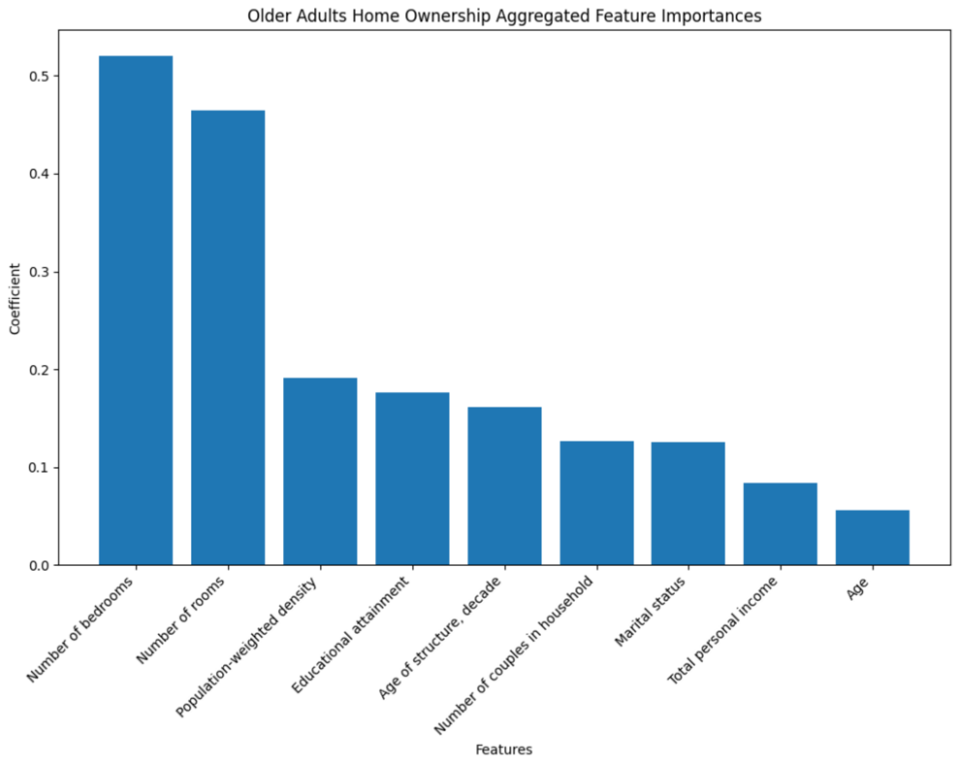


Figure 8. Aggregated feature importances for Older Adults Home Ownership.

Discussion

Young Adults Home Ownership

Young Adults are defined as adults aged 18 to 34. All SVCs performed similarly regardless of kernel. By pure magnitude, the top 5 predictors to predict homeownership in order are: “Number of bedrooms”, “Educational attainment”, “Total personal income”, “Number of rooms”, and “Population-weighted density.”

In particular, “Number of bedrooms” is by far the strongest predictor, with more bedrooms suggesting increased likelihood of homeownership. “Educational attainment” is a mixed bag; only a bachelor’s or a graduate degree suggest increased likelihood of homeownership, while the rest suggest increased likelihood of renting. “Total personal income” is the second most-positively correlated standalone feature; higher income suggests increased likelihood of homeownership. “Number of rooms” is like “Number of bedrooms” but with a much lower magnitude. Finally, “Population-weighted density” is negatively correlated with home ownership, meaning that a home in a denser area is more likely to be rented.

Middle-Aged Adults Home Ownership

Middle-Aged Adults are defined as adults aged 35 to 64. All SVCs performed similarly regardless of kernel. By pure magnitude, the top 5 predictors to predict homeownership in order are: “Number of bedrooms”, “Total personal income”, “Educational attainment”, “Age” (within the group), and “Number of rooms.”

There are similarities here to factors related with Young Adult homeownership – “Number of bedrooms”, “Total personal income”, “Educational attainment” are also its top 3, with “Number of bedrooms” towering above the rest, though income’s and education’s order is switched. The direction of the relationship hasn’t changed either – positive for all including “Age”. One thing to note here is that the age range is wider than the Young Adults age group. As a person grows older, they are more likely to own a home.

Older Adults Home Ownership

Older Adults are defined as adults aged 65+. All SVCs performed similarly regardless of kernel. By pure magnitude, the top 5 predictors to predict homeownership in order are: “Number of bedrooms”, “Number of rooms”, “Population-weighted density”, “Educational attainment”, and “Age of structure”.

Here we see a major break from previous trends. “Number of bedrooms” and “Number of rooms” are almost equal in magnitude. “Population-weighted density” and “Age of structure” (older homes are more likely to be rented) are now more important, while “Educational attainment” and “Total personal income” took a backseat. The direction of the remaining relationships remains the same.

Conclusion

This analysis aimed to assess the shift in feature importances for predicting homeownership across various age brackets: Young Adults, Middle-Aged Adults, and Older Adults. By employing Support Vector Classifiers with linear, radial, and polynomial kernels (tuned through 5-fold grid search cross validation), clear trends were uncovered.

For example, while “Number of bedrooms” remained a staple across all groups, “Educational attainment” and “Total personal income” slowly became less important as a person grows older. On the other hand, while “Population-weighted density” is a strong predictor for Young and Older Adults, it is at its weakest when used to predict for Middle-Aged Adults (with “Marital Status” overtaking its spot).

These are indicators of a changing time, homes were likely cheaper decades ago while income and purchasing power was comparatively higher, so personal income make less of a difference within the older generation. For the younger generation born in a more competitive housing market, “Educational attainment” and “Total personal income” (which are likely to have some correlation) become very important factors. Future analysis could attempt finer age group segmentations so that a smoother and more interpretable comparisons can be drawn.

References

1. Homeownership rate in the United States from 1990 to 2023 [Internet]. Statista Research Department; 2024 [cited 2024 Apr 29]. Available from: https://www.statista.com/statistics/184902/homeownership-rate-in-the-us-since-2003/
2. Goodman L, Choi JH, Zhu J. The “real” homeownership gap between today’s young adults and past generations is much larger than you think [Internet]. 2023 [cited 2024 Apr 29]. Available from: https://www.urban.org/urban-wire/real-homeownership-gap-between-todays-young-adults-and-past-generations-much-larger-you
3. Steven Ruggles, Sarah Flood, Matthew Sobek, Danika Brockman, Grace Cooper, Stephanie Richards, and Megan Schouweiler. IPUMS USA: Version 13.0 [dataset]. Minneapolis, MN: IPUMS, 2023. https://doi.org/10.18128/D010.V13.0
4. Choi J, Zhu J, Goodman L, Ganesh B, Strochak S. Millennial Homeownership [Internet]. 2018 [cited 2024 Apr 29]. Available from: https://www.urban.org/sites/default/files/publication/98729/millennial_homeownership_0.pdf
5. Choi J, Zhu J, Goodman L, Ganesh B, Strochak S [Internet]. 2018 [cited 2024 Apr 29]. Available from: https://www.urban.org/sites/default/files/publication/98729/millennial_homeownership_0.pdf
6. Sotelo D. Effect of Feature Standardization on Linear Support Vector Machines [Internet]. Towards Data Science; 2017 [cited 2024 Apr 29]. Available from: https://towardsdatascience.com/effect-of-feature-standardization-on-linear-support-vector-machines-13213765b812
7. Search for variables [Internet]. [cited 2024 Apr 29]. Available from: https://usa.ipums.org/usa-action/variables/live_search
8. IPUMS USA: DESCR: INCTOT [Internet]. [cited 2024 Apr 29]. Available from: https://usa.ipums.org/usa-action/variables/INCTOT#codes_section

Appendix

hw2

April 29, 2024

```
[ ]: import pandas as pd
     import numpy as np
     from sklearn.model_selection import KFold, train_test_split, GridSearchCV
     from sklearn.svm import SVC
     from sklearn.preprocessing import StandardScaler, LabelEncoder
     import matplotlib.pyplot as plt
     import matplotlib.patches as mpatches
     from ISLP.svm import plot as plot_svm
```

0.1 Data Preparation

```
[ ]: df = pd.read_csv("Housing.csv")
```

```
[ ]: # retrieve labels from https://usa.ipums.org/usa-action/variables/live_search␣
      ↪for exploration purposes
     col_labels = {
         "SERIAL": "Household serial number",
         "DENSITY": "Population-weighted density of PUMA",  # PUMA is Public Use␣
      ↪Microdata Area
         "OWNERSHP": "Ownership of dwelling (tenure) [general version]",  # owned/
      ↪being bought vs rented
         "OWNERSHPD": "Ownership of dwelling (tenure) [detailed version]",  # e.g.␣
      ↪rented vs no cash rent vs cash rent
         "COSTELEC": "Annual electricity cost",
         "COSTGAS": "Annual gas cost",
         "COSTWATR": "Annual water cost",
         "COSTFUEL": "Annual home heating fuel cost",
         "HHINCOME": "Total household income",
         "VALUEH": "House value",
         "ROOMS": "Number of rooms",
         "BUILTYR2": "Age of structure, decade",
         "BEDROOMS": "Number of bedrooms",
         "VEHICLES": "Vehicles available",
         "NFAMS": "Number of families in household",
         "NCOUPLES": "Number of couples in household",
         "PERNUM": "Person number in sample unit",
    "PERWT": "Person weight",  # sampling weight, indicates how many persons in␣
↪the U.S. population are represented by a given person
    "AGE": "Age",
    "MARST": "Marital status",
    "BIRTHYR": "Year of birth",
    "EDUC": "Educational attainment [general version]",
    "EDUCD": "Educational attainment [detailed version]",
    "INCTOT": "Total personal income",
}

df_readable = df.rename(columns=col_labels)
```

[ ]:
```
# filter by columns of interest and replace special missing values with NaN
cols_of_interest = [
    "Household serial number",
    "Population-weighted density of PUMA",
    "Ownership of dwelling (tenure) [general version]",
    "Number of rooms",
    "Age of structure, decade",
    "Number of bedrooms",
    "Number of families in household",
    "Number of couples in household",
    "Age",
    "Marital status",
    "Educational attainment [general version]",
    "Total personal income",
]

missing_values_per_col = {
    "Total personal income": [9999999],
}

df_filtered = df_readable[cols_of_interest].replace(missing_values_per_col, np.
↪nan)
```

[ ]:
```
# rename some long column names
cols_rename_map = {
    "Population-weighted density of PUMA": "Population-weighted density",
    "Ownership of dwelling (tenure) [general version]": "Ownership of dwelling",
    "Educational attainment [general version]": "Educational attainment",
}

df_filtered = df_filtered.rename(columns=cols_rename_map)
```

[ ]:
```
# filter to single family households with 0 or 1 couples -- 2 or 3 couples of␣
↪the same family living in the same household is an outlier.
df_single_family = df_filtered[
    (df_filtered["Number of families in household"] == 1)
    & (
        (df_filtered["Number of couples in household"] == 1)
        | (df_filtered["Number of couples in household"] == 0)
    )
]

# remove all rows with missing values
initial_length = len(df_single_family)
df_single_family = df_single_family.dropna()
new_length = len(df_single_family)
rows_dropped = initial_length - new_length
print(f"Removed {rows_dropped} rows with missing values out of {initial_length}␣
 ↪({(rows_dropped / initial_length)*100:.2f}%)")

# aggregate per head of household, defined as a person 18 or older with highest␣
 ↪personal income
index = (
    df_single_family[df_single_family["Age"] >= 18]
    .groupby("Household serial number")["Total personal income"]
    .idxmax()
)
df_single_family = df_single_family.loc[index]

# we can also aggregate by sorting, but seems like a bad approach
# df_single_family = df_single_family.sort_values("Total personal income",␣
 ↪ascending=False).drop_duplicates("Household serial number")

# drop Number of families in household since it's always 1
# drop Household serial number since we've already aggregated, and it's not␣
 ↪useful as a predictor
df_single_family = df_single_family.drop(columns=["Number of families in␣
 ↪household", "Household serial number"])
```

```
Removed 11840 rows with missing values out of 68131 (17.38%)
```

```
[ ]: # group education into 5 categories: less than high school, high school, some␣
      ↪college, bachelor's, graduate
     # by default is (], left-exclusive and right-inclusive
     bins = [0, 5, 6, 9, 10, 11]
     labels = [
         "Less than high school",
         "High school",
         "Some college",
         "Bachelor's",
         "Graduate",
     ]
```

```
df_single_family["Educational attainment"] = pd.cut(
    df_single_family["Educational attainment"],
    bins=bins,
    labels=labels,
    include_lowest=True,  # so 0 is included in 0-5
)
```

[ ]:
```
# group marital status into 3 categories: married (spouse present/absent),␣
↪previously married (separated/divorced/widowed), never married
bins = [1, 2, 5, 6]
labels = [
    "Married",
    "Previously married",
    "Never married",
]
df_single_family["Marital status"] = pd.cut(
    df_single_family["Marital status"],
    bins=bins,
    labels=labels,
    include_lowest=True,
)
```

[ ]:
```
# rename ownership of dwelling categories, 1 is Owned or being bought, 2 is␣
↪Rented
df_single_family = df_single_family.replace(
    {"Ownership of dwelling": {1: "Owned", 2: "Rented"}}
)
```

[ ]:
```
# encode categorical predictors as nominal, the distance betweeen our␣
↪categories is hard to quantify and may not be equal
# e.g. would distance between "Less than high school" and "High school" be the␣
↪same as "High school" and "Some college"?
nominal_cols = [
    "Educational attainment",
    "Marital status",
]

# one-hot encode (which is different to dummy encode) - doesn't drop first␣
↪category, easier to interpret and SVM can handle it
df_onehot = pd.get_dummies(df_single_family[nominal_cols], drop_first=False)
df_single_family = pd.concat([df_single_family, df_onehot], axis=1)
df_single_family = df_single_family.drop(columns=nominal_cols)
```

[ ]:
```
# group age into 3 categories: 18-34 (young adults), 35-64 (middle-aged␣
↪adults), 65+ (older adults)
bins = [18, 34, 64, float('inf')]
labels = ["Young adults", "Middle-aged adults", "Older adults"]
```

```python
# keep age as original data for later
age_groups = pd.cut(
    df_single_family["Age"],
    bins=bins,
    labels=labels,
    include_lowest=True
)

# create 3 separate dataframes for each age group
df_young_adults = df_single_family[age_groups == "Young adults"]
df_middle_aged_adults = df_single_family[age_groups == "Middle-aged adults"]
df_older_adults = df_single_family[age_groups == "Older adults"]
```

[ ]:
```python
# scaler for later use
scaler = StandardScaler()
```

0.2 Modeling

0.2.1 Young Adults

[ ]:
```python
X = df_young_adults.drop(columns=['Ownership of dwelling'])
y = df_young_adults['Ownership of dwelling']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=1)
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Radial

[ ]:
```python
svm_rbf = SVC(kernel="rbf", tol=0.1)

kFold = KFold(n_splits=5, shuffle=True, random_state=1)
params = {"C": [0.001, 0.01, 0.1, 1, 10], "gamma": [0.0001, 0.001, 0.01, 1, 10,␣
 ↪100]}
cv = GridSearchCV(
    svm_rbf,
    param_grid=params,
    cv=kFold,
    refit=True,
    n_jobs=-1,
    scoring="accuracy",
    verbose=1,
)
cv.fit(X_train_scaled, y_train)

train_accuracy = cv.best_estimator_.score(X_train_scaled, y_train)
test_accuracy = cv.best_estimator_.score(X_test_scaled, y_test)
```

```
best_C_rbf = cv.best_params_["C"]
best_gamma = cv.best_params_["gamma"]
print(f"Radial CV results: C={best_C_rbf}, gamma={best_gamma}")
print(f"Train accuracy: {train_accuracy*100:.2f}%")
print(f"Test accuracy: {test_accuracy*100:.2f}%")
```

```
Fitting 5 folds for each of 30 candidates, totalling 150 fits
Radial CV results: C=10, gamma=0.01
Train accuracy: 80.52%
Test accuracy: 79.76%
```

Poly

[ ]:
```
svm_poly = SVC(kernel="poly", tol=0.1)

kFold = KFold(n_splits=5, shuffle=True, random_state=1)
params = {
    "C": [0.001, 0.01, 0.1, 1, 10],
    "degree": np.arange(2, 5, 1),
    "coef0": np.arange(0, 3, 0.5),
}
cv = GridSearchCV(
    svm_poly,
    param_grid=params,
    cv=kFold,
    refit=True,
    n_jobs=-1,
    scoring="accuracy",
    verbose=1,
)
cv.fit(X_train_scaled, y_train)

train_accuracy = cv.best_estimator_.score(X_train_scaled, y_train)
test_accuracy = cv.best_estimator_.score(X_test_scaled, y_test)

best_C_poly = cv.best_params_["C"]
best_degree = cv.best_params_["degree"]
best_coef0 = cv.best_params_["coef0"]

print(f"Poly CV results: C={best_C_poly}, degree={best_degree},␣
 ↪coef0={best_coef0}")
print(f"Train accuracy: {train_accuracy*100:.2f}%")
print(f"Test accuracy: {test_accuracy*100:.2f}%")
```

```
Fitting 5 folds for each of 90 candidates, totalling 450 fits
Poly CV results: C=0.1, degree=2, coef0=1.5
Train accuracy: 80.19%
Test accuracy: 79.69%
```

Linear

```
[ ]: svm_linear = SVC(kernel="linear", tol=0.1)

     kFold = KFold(n_splits=5, shuffle=True, random_state=1)
     params = {"C": [0.001, 0.01, 0.1, 1, 10]}
     cv = GridSearchCV(
         svm_linear,
         param_grid=params,
         cv=kFold,
         refit=True,
         n_jobs=-1,
         scoring="accuracy",
         verbose=1,
     )
     cv.fit(X_train_scaled, y_train)

     train_accuracy = cv.best_estimator_.score(X_train_scaled, y_train)
     test_accuracy = cv.best_estimator_.score(X_test_scaled, y_test)

     best_C_linear = cv.best_params_["C"]

     print(f"Linear CV results: C={best_C_linear}")
     print(f"Train accuracy: {train_accuracy*100:.2f}%")
     print(f"Test accuracy: {test_accuracy*100:.2f}%")
```

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits
Linear CV results: C=0.1
Train accuracy: 78.96%
Test accuracy: 79.20%
```

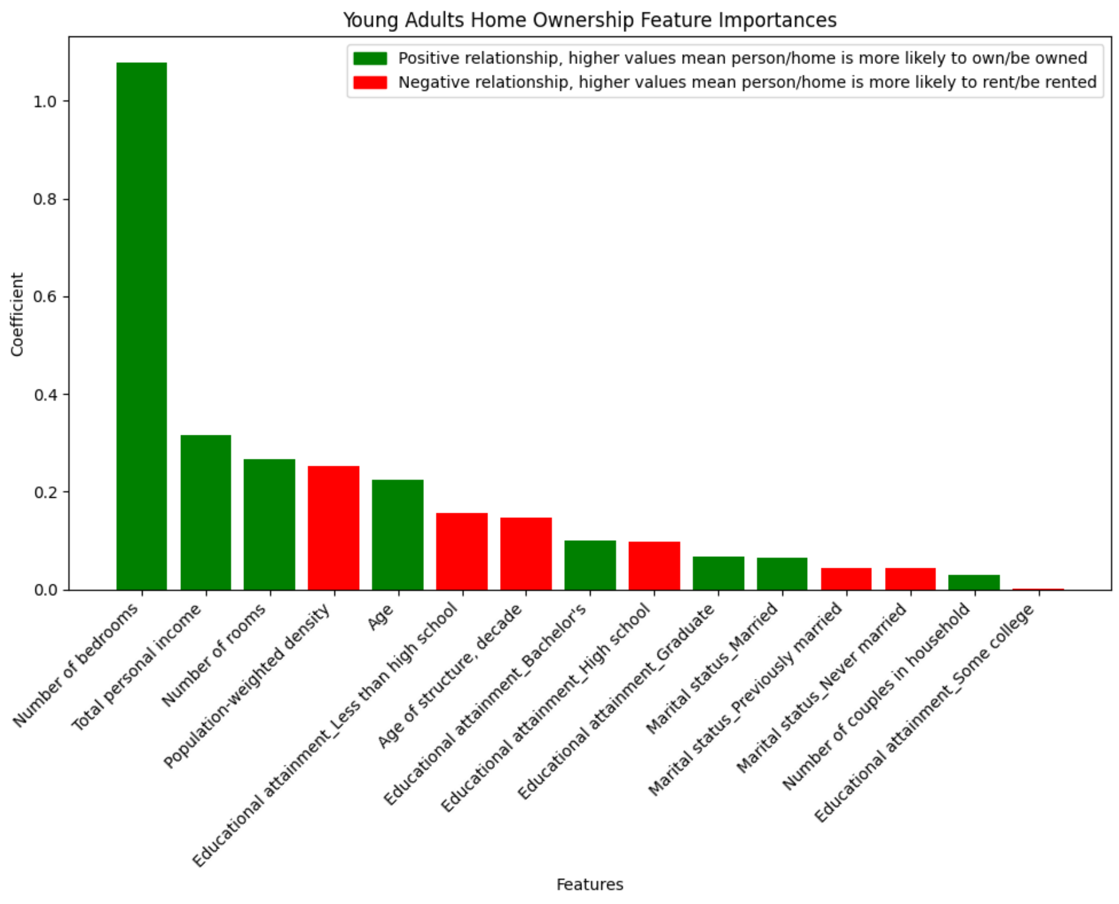
Feature Importance

```
[ ]: coef = cv.best_estimator_.coef_[0]  # using linear kernel
     feature_importances = pd.DataFrame({"Feature": X.columns, "Coefficient": coef,␣
      ↪"AbsCoefficient": abs(coef)})
     sorted_importances = feature_importances.sort_values(by="AbsCoefficient",␣
      ↪ascending=False)

     # sum up importances of onehot encoded features
     grouped_importances = feature_importances.groupby(
         feature_importances["Feature"].str.split("_").str[0]
     ).sum()
     sorted_grouped_importances = grouped_importances.
      ↪sort_values(by="AbsCoefficient", ascending=False)
```

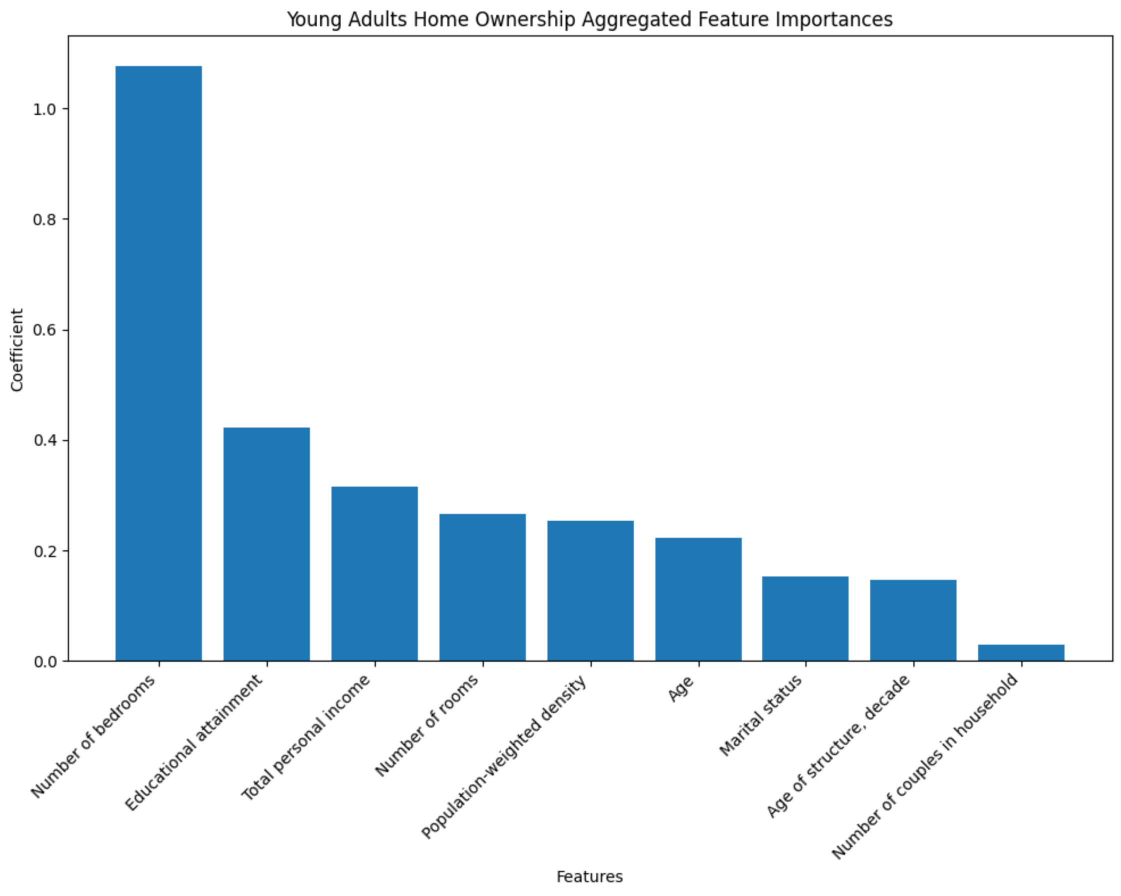
```
/var/folders/jy/pdgtcrw968d_3_tqfzr3w6y00000gn/T/ipykernel_84534/2379649311.py:6
: FutureWarning: The default value of numeric_only in DataFrameGroupBy.sum is
deprecated. In a future version, numeric_only will default to False. Either
specify numeric_only or select only columns which should be valid for the
```

```
function.
  grouped_importances = feature_importances.groupby(
```

[ ]:
```
# color the bars based on coefficient sign
colors = ['green' if coef < 0 else 'red' for coef in␣
 ↪sorted_importances["Coefficient"]]
green_patch = mpatches.Patch(color='green', label='Positive relationship,␣
 ↪higher values mean person/home is more likely to own/be owned')
red_patch = mpatches.Patch(color='red', label='Negative relationship, higher␣
 ↪values mean person/home is more likely to rent/be rented')

plt.figure(figsize=(10, 8))
plt.bar(sorted_importances['Feature'], sorted_importances["AbsCoefficient"],␣
 ↪color=colors)
plt.legend(handles=[green_patch, red_patch])
plt.xlabel("Features")
plt.ylabel("Coefficient")
plt.title("Young Adults Home Ownership Feature Importances")
plt.xticks(rotation=45, ha="right")
plt.tight_layout()
plt.show()
```



[ ]:
```
plt.figure(figsize=(10, 8))
plt.bar(sorted_grouped_importances.index,␣
 ↪sorted_grouped_importances["AbsCoefficient"])
plt.xlabel("Features")
plt.ylabel("Coefficient")
plt.title("Young Adults Home Ownership Aggregated Feature Importances")
plt.xticks(rotation=45, ha="right")
plt.tight_layout()
plt.show()
```



0.2.2 Middle-Aged Adults

[ ]:
```
X = df_middle_aged_adults.drop(columns=['Ownership of dwelling'])
y = df_middle_aged_adults['Ownership of dwelling']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=1)
```

```
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Radial

```
[ ]: svm_rbf = SVC(kernel="rbf", tol=0.1)

     kFold = KFold(n_splits=5, shuffle=True, random_state=1)
     params = {"C": [0.001, 0.01, 0.1, 1, 10], "gamma": [0.0001, 0.001, 0.01, 1, 10,␣
       ↪100]}
     cv = GridSearchCV(
         svm_rbf,
         param_grid=params,
         cv=kFold,
         refit=True,
         n_jobs=-1,
         scoring="accuracy",
         verbose=1,
     )
     cv.fit(X_train_scaled, y_train)

     train_accuracy = cv.best_estimator_.score(X_train_scaled, y_train)
     test_accuracy = cv.best_estimator_.score(X_test_scaled, y_test)

     best_C_rbf = cv.best_params_["C"]
     best_gamma = cv.best_params_["gamma"]
     print(f"Radial CV results: C={best_C_rbf}, gamma={best_gamma}")
     print(f"Train accuracy: {train_accuracy*100:.2f}%")
     print(f"Test accuracy: {test_accuracy*100:.2f}%")
```

```
Fitting 5 folds for each of 30 candidates, totalling 150 fits

/opt/homebrew/anaconda3/envs/py39/lib/python3.9/site-
packages/joblib/externals/loky/process_executor.py:752: UserWarning: A worker
stopped while some jobs were given to the executor. This can be caused by a too
short worker timeout or by a memory leak.
  warnings.warn(

Radial CV results: C=10, gamma=0.001
Train accuracy: 82.09%
Test accuracy: 82.59%
```

Poly

```
[ ]: svm_poly = SVC(kernel="poly", tol=0.1)

     kFold = KFold(n_splits=5, shuffle=True, random_state=1)
     params = {
         "C": [0.001, 0.01, 0.1, 1, 10],
         "degree": np.arange(2, 5, 1),
              "coef0": np.arange(0, 3, 0.5),
          }
          cv = GridSearchCV(
              svm_poly,
              param_grid=params,
              cv=kFold,
              refit=True,
              n_jobs=-1,
              scoring="accuracy",
              verbose=1,
          )
          cv.fit(X_train_scaled, y_train)

          train_accuracy = cv.best_estimator_.score(X_train_scaled, y_train)
          test_accuracy = cv.best_estimator_.score(X_test_scaled, y_test)

          best_C_poly = cv.best_params_["C"]
          best_degree = cv.best_params_["degree"]
          best_coef0 = cv.best_params_["coef0"]

          print(f"Poly CV results: C={best_C_poly}, degree={best_degree},␣
            ↪coef0={best_coef0}")
          print(f"Train accuracy: {train_accuracy*100:.2f}%")
          print(f"Test accuracy: {test_accuracy*100:.2f}%")
```

```
Fitting 5 folds for each of 90 candidates, totalling 450 fits
```

```
/opt/homebrew/anaconda3/envs/py39/lib/python3.9/site-
packages/joblib/externals/loky/process_executor.py:752: UserWarning: A worker
stopped while some jobs were given to the executor. This can be caused by a too
short worker timeout or by a memory leak.
  warnings.warn(
```

```
Poly CV results: C=10, degree=2, coef0=0.5
Train accuracy: 82.31%
Test accuracy: 82.87%
```

Linear

```
[ ]: svm_linear = SVC(kernel="linear", tol=0.1)

     kFold = KFold(n_splits=5, shuffle=True, random_state=1)
     params = {"C": [0.001, 0.01, 0.1, 1, 10]}
     cv = GridSearchCV(
         svm_linear,
         param_grid=params,
         cv=kFold,
         refit=True,
         n_jobs=-1,
    scoring="accuracy",
    verbose=1,
)
cv.fit(X_train_scaled, y_train)

train_accuracy = cv.best_estimator_.score(X_train_scaled, y_train)
test_accuracy = cv.best_estimator_.score(X_test_scaled, y_test)

best_C_linear = cv.best_params_["C"]

print(f"Linear CV results: C={best_C_linear}")
print(f"Train accuracy: {train_accuracy*100:.2f}%")
print(f"Test accuracy: {test_accuracy*100:.2f}%")
```

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits
```

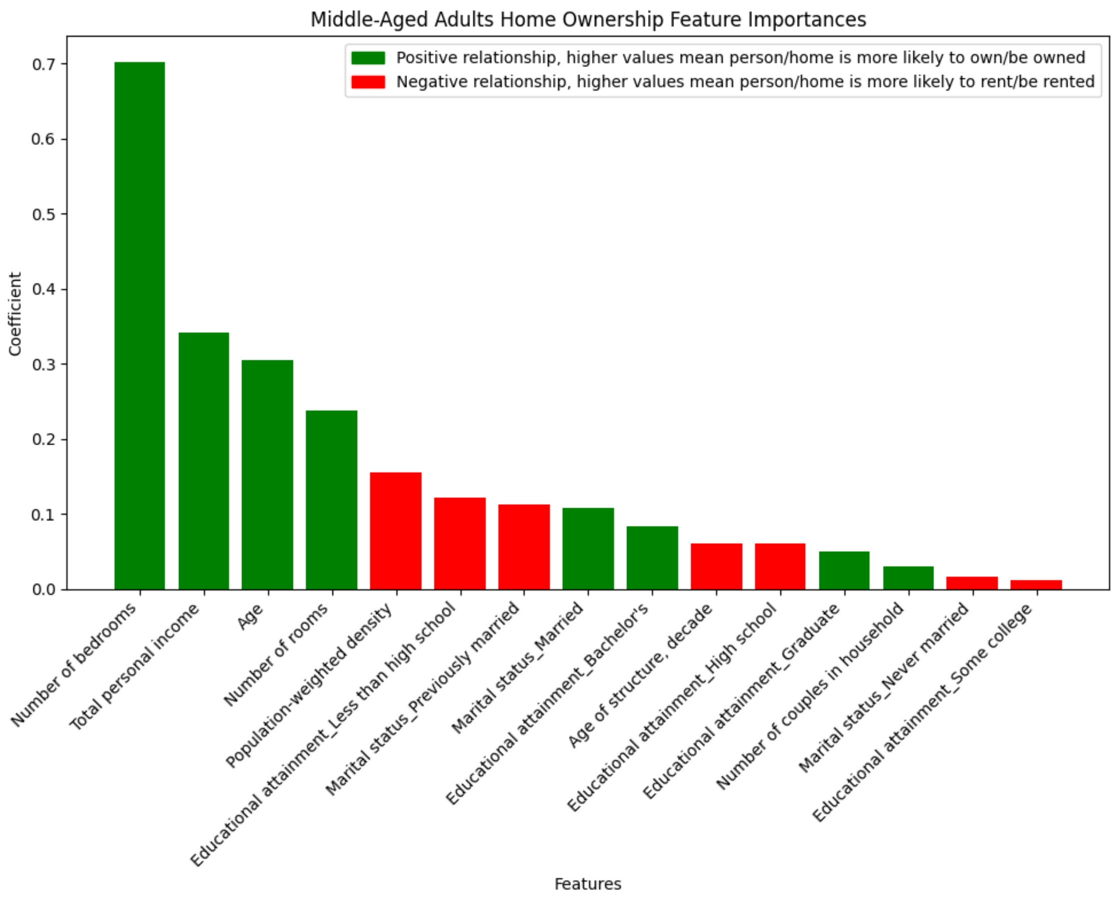
```
/opt/homebrew/anaconda3/envs/py39/lib/python3.9/site-
packages/joblib/externals/loky/process_executor.py:752: UserWarning: A worker
stopped while some jobs were given to the executor. This can be caused by a too
short worker timeout or by a memory leak.
  warnings.warn(
```

```
Linear CV results: C=1
Train accuracy: 82.22%
Test accuracy: 82.61%
```

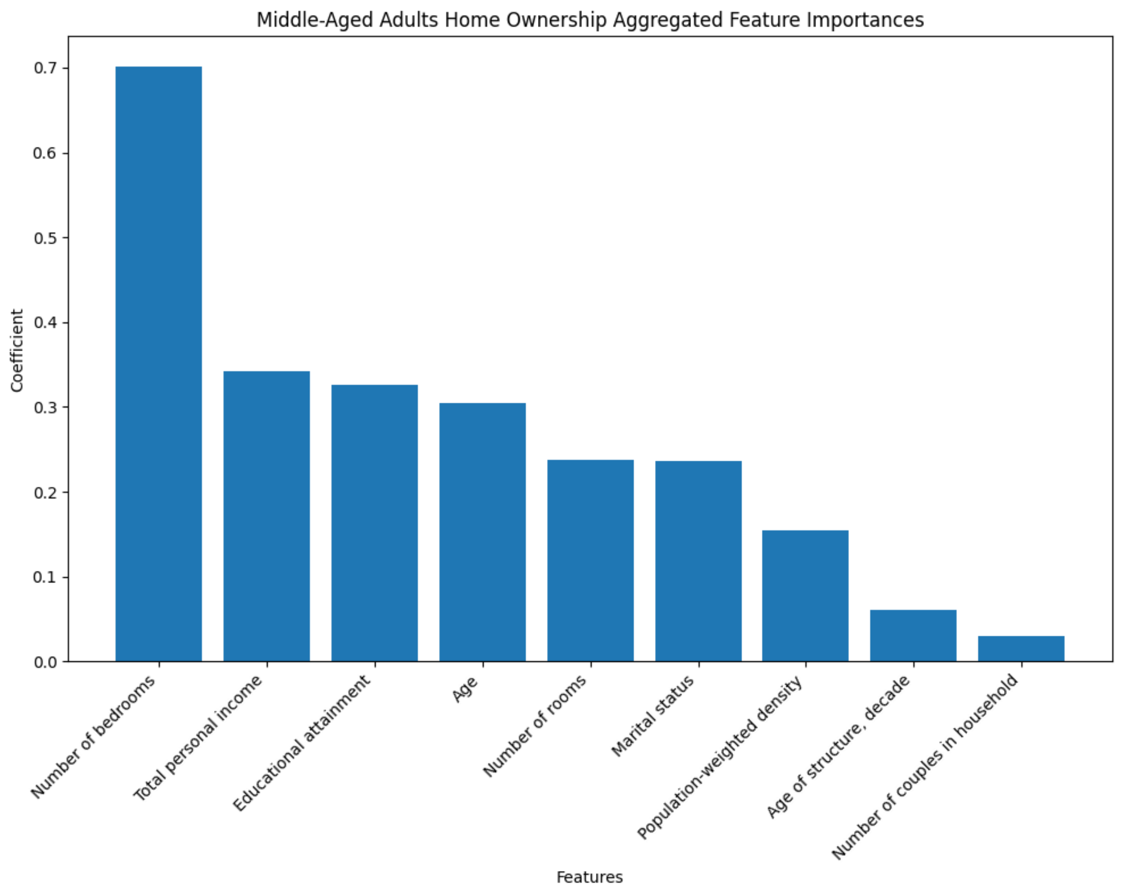
Feature Importance

[ ]:
```
coef = cv.best_estimator_.coef_[0]  # using linear kernel
feature_importances = pd.DataFrame({"Feature": X.columns, "Coefficient": coef,␣
 ↪"AbsCoefficient": abs(coef)})
sorted_importances = feature_importances.sort_values(by="AbsCoefficient",␣
 ↪ascending=False)

# sum up importances of onehot encoded features
grouped_importances = feature_importances.groupby(
    feature_importances["Feature"].str.split("_").str[0]
).sum()
sorted_grouped_importances = grouped_importances.
 ↪sort_values(by="AbsCoefficient", ascending=False)
```

```
/var/folders/jy/pdgtcrw968d_3_tqfzr3w6y00000gn/T/ipykernel_84534/2379649311.py:6
: FutureWarning: The default value of numeric_only in DataFrameGroupBy.sum is
deprecated. In a future version, numeric_only will default to False. Either
specify numeric_only or select only columns which should be valid for the
function.
  grouped_importances = feature_importances.groupby(
[ ]: # color the bars based on coefficient sign
     colors = ['green' if coef < 0 else 'red' for coef in␣
      ↪sorted_importances["Coefficient"]]
     green_patch = mpatches.Patch(color='green', label='Positive relationship,␣
      ↪higher values mean person/home is more likely to own/be owned')
     red_patch = mpatches.Patch(color='red', label='Negative relationship, higher␣
      ↪values mean person/home is more likely to rent/be rented')

     plt.figure(figsize=(10, 8))
     plt.bar(sorted_importances['Feature'], sorted_importances["AbsCoefficient"],␣
      ↪color=colors)
     plt.legend(handles=[green_patch, red_patch])
     plt.xlabel("Features")
     plt.ylabel("Coefficient")
     plt.title("Middle-Aged Adults Home Ownership Feature Importances")
     plt.xticks(rotation=45, ha="right")
     plt.tight_layout()
     plt.show()
```



```
[ ]: plt.figure(figsize=(10, 8))
     plt.bar(sorted_grouped_importances.index,␣
      ↪sorted_grouped_importances["AbsCoefficient"])
     plt.xlabel("Features")
     plt.ylabel("Coefficient")
     plt.title("Middle-Aged Adults Home Ownership Aggregated Feature Importances")
     plt.xticks(rotation=45, ha="right")
     plt.tight_layout()
     plt.show()
```



0.2.3 Older Adults

```
[ ]: X = df_older_adults.drop(columns=['Ownership of dwelling'])
     y = df_older_adults['Ownership of dwelling']
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
      ↪random_state=1)
     X_train_scaled = scaler.fit_transform(X_train)
     X_test_scaled = scaler.transform(X_test)
```

Radial

```
[ ]: svm_rbf = SVC(kernel="rbf", tol=0.1)

     kFold = KFold(n_splits=5, shuffle=True, random_state=1)
     params = {"C": [0.001, 0.01, 0.1, 1, 10], "gamma": [0.0001, 0.001, 0.01, 1, 10,␣
      ↪100]}
     cv = GridSearchCV(
         svm_rbf,
         param_grid=params,
         cv=kFold,
         refit=True,
         n_jobs=-1,
         scoring="accuracy",
         verbose=1,
     )
     cv.fit(X_train_scaled, y_train)

     train_accuracy = cv.best_estimator_.score(X_train_scaled, y_train)
     test_accuracy = cv.best_estimator_.score(X_test_scaled, y_test)

     best_C_rbf = cv.best_params_["C"]
     best_gamma = cv.best_params_["gamma"]
     print(f"Radial CV results: C={best_C_rbf}, gamma={best_gamma}")
     print(f"Train accuracy: {train_accuracy*100:.2f}%")
     print(f"Test accuracy: {test_accuracy*100:.2f}%")
```

```
Fitting 5 folds for each of 30 candidates, totalling 150 fits

/opt/homebrew/anaconda3/envs/py39/lib/python3.9/site-
packages/joblib/externals/loky/process_executor.py:752: UserWarning: A worker
stopped while some jobs were given to the executor. This can be caused by a too
short worker timeout or by a memory leak.
  warnings.warn(

Radial CV results: C=10, gamma=0.01
Train accuracy: 88.07%
Test accuracy: 87.06%
```

Poly

```
[ ]: svm_poly = SVC(kernel="poly", tol=0.1)

     kFold = KFold(n_splits=5, shuffle=True, random_state=1)
     params = {
         "C": [0.001, 0.01, 0.1, 1, 10],
         "degree": np.arange(2, 5, 1),
         "coef0": np.arange(0, 3, 0.5),
     }
     cv = GridSearchCV(
    svm_poly,
    param_grid=params,
    cv=kFold,
    refit=True,
    n_jobs=-1,
    scoring="accuracy",
    verbose=1,
)
cv.fit(X_train_scaled, y_train)

train_accuracy = cv.best_estimator_.score(X_train_scaled, y_train)
test_accuracy = cv.best_estimator_.score(X_test_scaled, y_test)

best_C_poly = cv.best_params_["C"]
best_degree = cv.best_params_["degree"]
best_coef0 = cv.best_params_["coef0"]

print(f"Poly CV results: C={best_C_poly}, degree={best_degree},␣
  ↪coef0={best_coef0}")
print(f"Train accuracy: {train_accuracy*100:.2f}%")
print(f"Test accuracy: {test_accuracy*100:.2f}%")
```

```
Fitting 5 folds for each of 90 candidates, totalling 450 fits
Poly CV results: C=1, degree=3, coef0=1.0
Train accuracy: 88.58%
Test accuracy: 87.06%
```

Linear

[ ]:
```
svm_linear = SVC(kernel="linear", tol=0.1)

kFold = KFold(n_splits=5, shuffle=True, random_state=1)
params = {"C": [0.001, 0.01, 0.1, 1, 10]}
cv = GridSearchCV(
    svm_linear,
    param_grid=params,
    cv=kFold,
    refit=True,
    n_jobs=-1,
    scoring="accuracy",
    verbose=1,
)
cv.fit(X_train_scaled, y_train)

train_accuracy = cv.best_estimator_.score(X_train_scaled, y_train)
test_accuracy = cv.best_estimator_.score(X_test_scaled, y_test)

best_C_linear = cv.best_params_["C"]
```

```
print(f"Linear CV results: C={best_C_linear}")
print(f"Train accuracy: {train_accuracy*100:.2f}%")
print(f"Test accuracy: {test_accuracy*100:.2f}%")
```

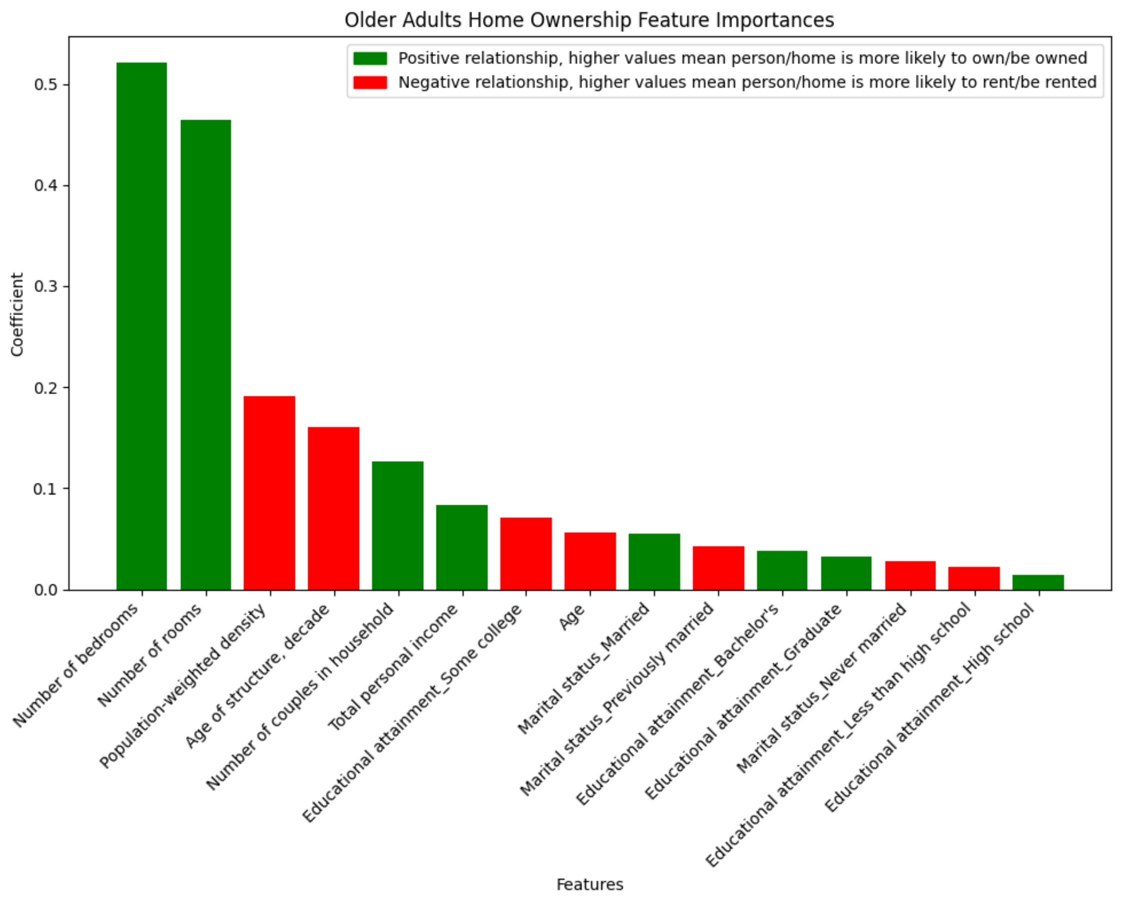
```
Fitting 5 folds for each of 5 candidates, totalling 25 fits
Linear CV results: C=10
Train accuracy: 87.58%
Test accuracy: 87.10%
```

Feature Importance

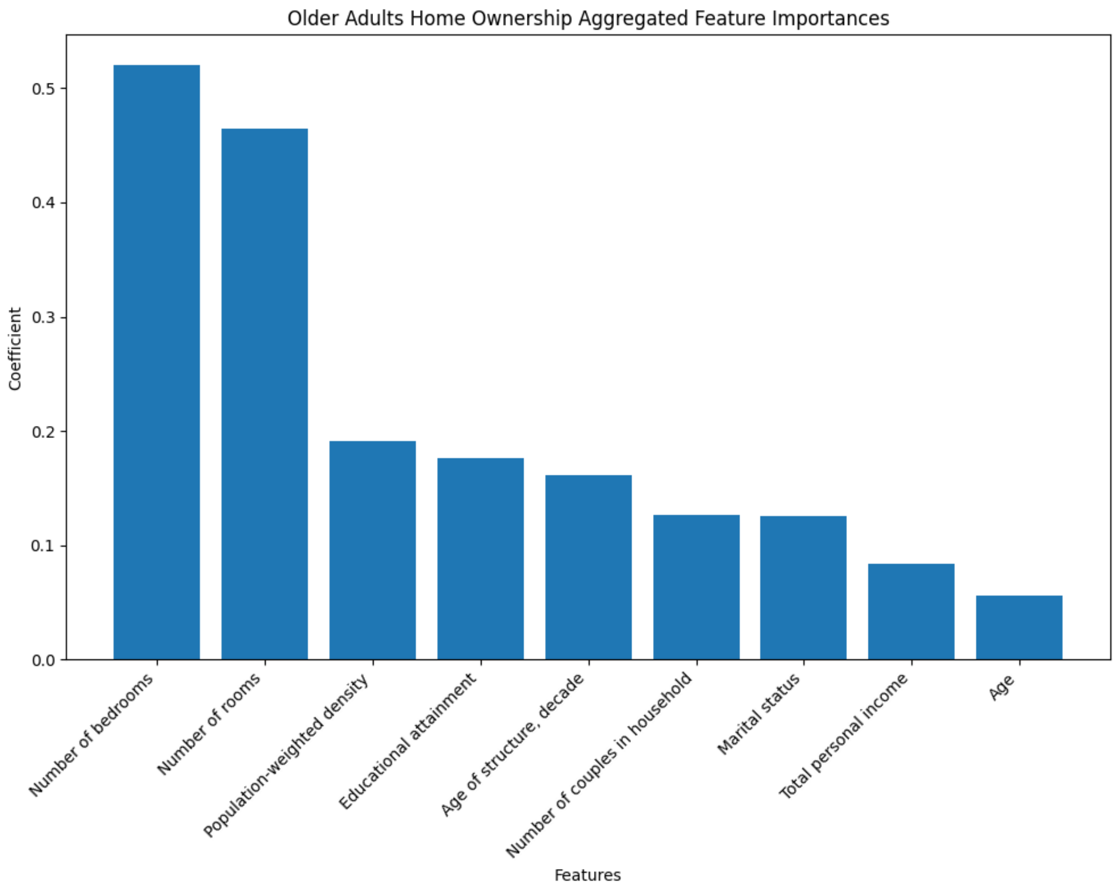
```
[ ]: coef = cv.best_estimator_.coef_[0]  # using linear kernel
feature_importances = pd.DataFrame({"Feature": X.columns, "Coefficient": coef,␣
 ↪"AbsCoefficient": abs(coef)})
sorted_importances = feature_importances.sort_values(by="AbsCoefficient",␣
 ↪ascending=False)

# sum up importances of onehot encoded features
grouped_importances = feature_importances.groupby(
    feature_importances["Feature"].str.split("_").str[0]
).sum()
sorted_grouped_importances = grouped_importances.
 ↪sort_values(by="AbsCoefficient", ascending=False)
```

```
/var/folders/jy/pdgtcrw968d_3_tqfzr3w6y00000gn/T/ipykernel_84534/2379649311.py:6
: FutureWarning: The default value of numeric_only in DataFrameGroupBy.sum is
deprecated. In a future version, numeric_only will default to False. Either
specify numeric_only or select only columns which should be valid for the
function.
  grouped_importances = feature_importances.groupby(
```

```
[ ]: # color the bars based on coeffficient sign
colors = ['green' if coef < 0 else 'red' for coef in␣
 ↪sorted_importances["Coefficient"]]
green_patch = mpatches.Patch(color='green', label='Positive relationship,␣
 ↪higher values mean person/home is more likely to own/be owned')
red_patch = mpatches.Patch(color='red', label='Negative relationship, higher␣
 ↪values mean person/home is more likely to rent/be rented')

plt.figure(figsize=(10, 8))
plt.bar(sorted_importances['Feature'], sorted_importances["AbsCoefficient"],␣
 ↪color=colors)
plt.legend(handles=[green_patch, red_patch])
plt.xlabel("Features")
plt.ylabel("Coefficient")
plt.title("Older Adults Home Ownership Feature Importances")
plt.xticks(rotation=45, ha="right")
plt.tight_layout()
plt.show()
```



```
[ ]: plt.figure(figsize=(10, 8))
     plt.bar(sorted_grouped_importances.index,␣
      ↪sorted_grouped_importances["AbsCoefficient"])
     plt.xlabel("Features")
     plt.ylabel("Coefficient")
     plt.title("Older Adults Home Ownership Aggregated Feature Importances")
     plt.xticks(rotation=45, ha="right")
     plt.tight_layout()
     plt.show()
```



0.3 Misc

0.3.1 Prove 'Owned' was encoded as class 0 (to determine direction of relationship, i.e. negative coef == more likely to own)

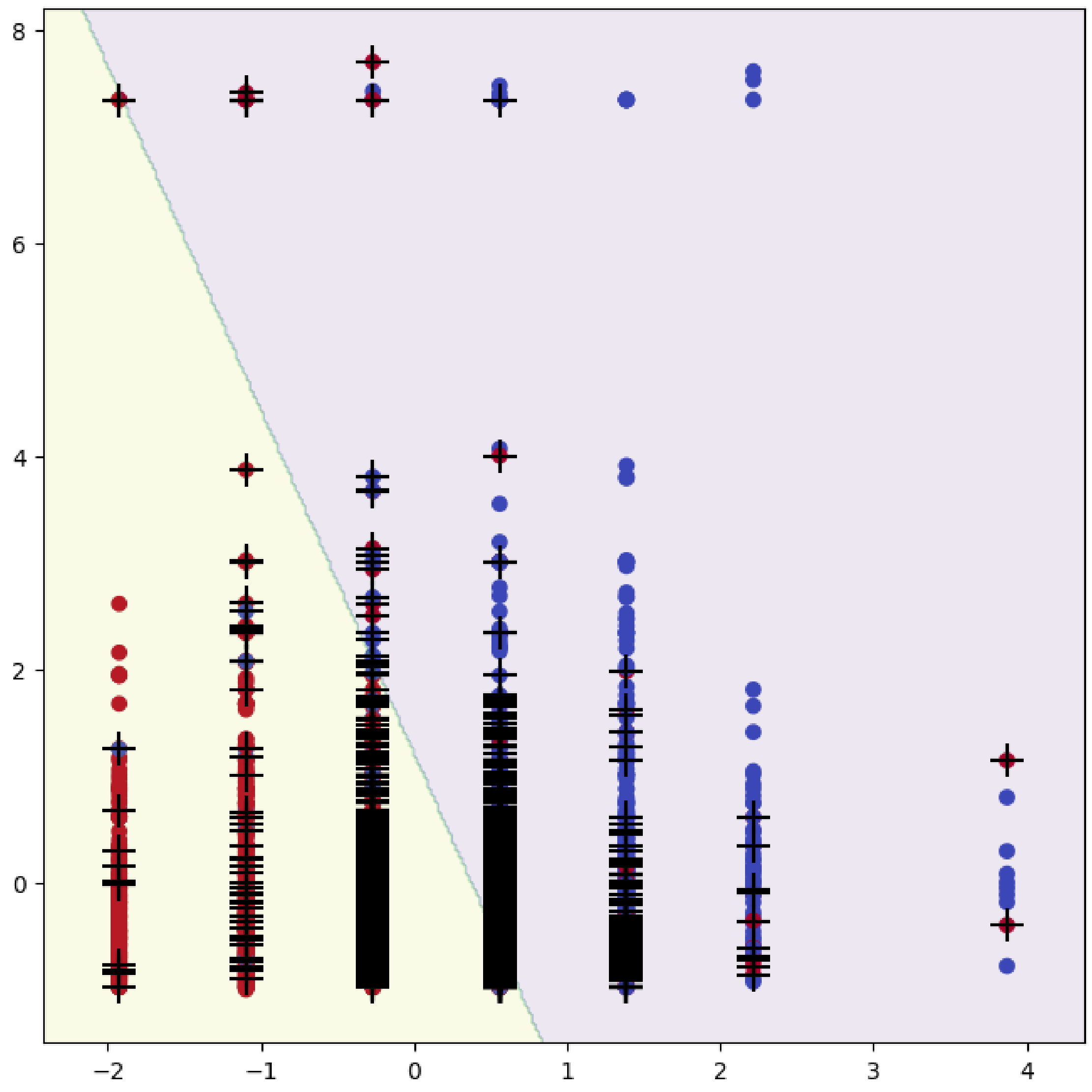
[ ]:
```
encoder = LabelEncoder()
encoder.fit_transform(y)

print(encoder.classes_)
```

```
['Owned' 'Rented']
```

0.3.2 'Cartoon' plot for a simplified model using 2 strong predictors

```
[ ]: X = df_young_adults[['Number of bedrooms', 'Total personal income']]
     y = df_young_adults['Ownership of dwelling']
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
      ↪random_state=1)
     X_train_scaled = scaler.fit_transform(X_train)
     X_test_scaled = scaler.transform(X_test)
```

```
[ ]: encoder = LabelEncoder()
     y_train_encoded = encoder.fit_transform(y_train)

     svm_linear = SVC(kernel="linear", C=0.01)
     svm_linear.fit(X_train_scaled, y_train_encoded)

     _, ax = plt.subplots(figsize=(8, 8))
     plot_svm(
         X_train_scaled,
         y_train_encoded,
         svm_linear,
         ax=ax,
         scatter_cmap=plt.cm.coolwarm,
         decision_cmap=plt.cm.viridis,
         alpha=0.1
     )
```


8
6
4
2
0
−2
−1
0
1
2
3
4