1. ABSTRACT

This project explores the application of Support Vector Machines (SVM) in predicting homeownership using demographic and economic data. SVM is a machine learning algorithm that can be used for both classification and regression tasks. The project discusses the mathematical formulation, kernel functions, and tuning parameters of SVM. The findings suggest that age of person, number of bedrooms in the house, cost of water and number of rooms are strong predictors of homeownership. The project also includes a plot of the SVM decision boundary to visualize the classifier's performance on two strong predictor variables. The implications of the findings are discussed in the context of current societal challenges surrounding access to housing, and suggestions are made for policymakers. This project demonstrates the potential of SVM in predicting homeownership and provides insights into the factors that influence homeownership.

2. INTRODUCTION

In this report, support vector machines (SVMs) are explored for the purpose of performing classification tasks. In this project, we will SVMS to investigate whether a home belongs to an owner or rented by a tenant based on several individuals and housing variables.

2.1. Dataset Description

The data used in this project is downloaded from United States Census Bureau (IPUMS USA). The dataset contains housing variables such as number of rooms, number of bedrooms, cost of water, cost of gas, cost of electricity etc, people variables such as income, age, marital status etc and many other variables. It has OWNERSHP variable which is a categorical variable that determines if a person is owner of the house(1) or renting the house(2). This is our target variable. We have a variable SERIAL that uniquely identifies an household and PERNUM that says how many people are there in an household. The dataset contains total 23 variables with 75388 observations

2.2 Problem Types

In this report, we will use support vector classifier to investigate the classification problem. In this, we will predict where a person is owner of a house or a renter based on several other factors such as age, income, marital status, number of rooms of house, costs of electricity, gas, water etc. We will demonstrate the use of support vectors, including different kernels, for this problem.

3. THEORETICAL BACKGROUND

3.1 Support Vector Machines (SVM)

They are a type of supervised learning algorithm used for classification tasks. The goal of an SVM is to find the best hyperplane that separates data points into different classes. The hyperplane is chosen to have the maximum margin, which is the distance between the hyperplane and the nearest data points from each class.

SVMs are known to be effective for high-dimensional data sets and can handle both linear and non-linearly separable data. SVMs use kernels to transform data into a higher-dimensional space where it is more easily separable. The kernel function calculates the similarity between two data points and measures the distance between them.

The three types of kernels we will investigate in this report are:

3.2 Linear kernel:

The linear kernel assumes that the data is linearly separable. It is the simplest and fastest kernel to compute, making it a good choice for large datasets with high dimensions.

3.3 Radial basis function (RBF) kernel:

The RBF kernel is used when the data is not linearly separable. It maps the data into a high-dimensional space using a Gaussian function and finds the hyperplane that separates the data points.

3.4 Polynomial kernel:

The polynomial kernel is used to find the decision boundary between data points that are not linearly separable. It maps the data into a higher-dimensional space using a polynomial function.

3.5 Tuning Parameters:

In addition to selecting the appropriate kernel, SVMs also require the selection of tuning parameters such as the regularization parameter C and the kernel parameter γ for the RBF

kernel. The regularization parameter C controls the trade-off between achieving a low training error and a low testing error. The kernel parameter γ controls the width of the kernel and affects the smoothness of the decision boundary.

Overall, SVMs are a powerful tool for classification tasks and can provide accurate results when properly tuned and applied.

4. METHODOLOGY

4.1. Data Cleaning

In this project, we aimed to build a Support Vector Machine (SVM) model to predict home ownership based on various demographic and socio-economic factors. To achieve this, we started by exploring the dataset, which contained 75,388 rows and 23 columns.

As the first step of data preprocessing, we decided to subset the data to only include elder person rows for each household. This resulted in a reduced dataset containing 30,802 rows and 23 columns.

Next, we identified columns that were not necessary for our analysis and dropped them from the dataset. These columns included SERIAL (unique number for each household), OWNERSHPD (detailed ownership status), PERNUM (person number in household), PERWT (person weight), BIRTHYR (person's birth year), EDUC (education attainment), and EDUCD (detailed education attainment). This cleaning step ensured that we were working with relevant data for our analysis.

We also checked for any missing values in the remaining dataset and found that there were none. After cleaning the data, we encoded the categorical variable MARST (marital status) using its labels.

To prepare the data for modeling, we split it into training and testing sets using the train_test_split() function from the sklearn library. Additionally, we applied scaling using the StandardScaler() function to ensure that all features were on the same scale. This step helped us avoid any bias in the model due to the differences in the ranges of the features.

Overall, by following these preprocessing steps, we were able to obtain a clean and relevant dataset that we could use to build and evaluate our SVM model.

4.2 SVM Using Linear Kernel:

The SVM model with linear kernel is trained using the above that. Then also performed feature selection using the SelectKBest algorithm from scikit-learn. It selects the top k features that have the highest score with respect to the target variable using the f_classif function, which is a statistical test that measures the difference in mean values of the features for each class of the target variable.

After performing feature selection, the code trains and evaluates a linear SVM classifier with different values of the regularization parameter C. It prints the accuracy score for each value of C.

Finally, calculated the feature importances using permutation importance and plotted the top 5 important features. Also created countplots for each of the top 5 features to visualize their relationship with the target variable. Then plotted a decision boundary graph using only top 2 predictor variables.

4.3 SVM Using Radial Kernel:

Similar to the linear kernel, I trained an SVM model with radial kernel and performed feature selection using SelectKBest algorithm. Then, I evaluated the model with different values of C and gamma. For each combination of C and gamma, I printed the accuracy score.

Finally, I calculated the feature importances using permutation importance and plotted the top 5 important features. I also created countplots for each of the top 5 features to visualize their relationship with the target variable. Then, I plotted a decision boundary graph using only top 2 predictor variables.

4.4 SVM Using Polynomial Kernel:

Similarly, I trained an SVM model with polynomial kernel and performed feature selection using SelectKBest algorithm. Then, I evaluated the model with different values of C and degree. For each combination of C and degree, I printed the accuracy score.

After that, I calculated the feature importances using permutation importance and plotted the top 5 important features. I also created countplots for each of the top 5 features to visualize their relationship with the target variable. Finally, I plotted a decision boundary graph using only top 2 predictor variables.

5. COMPUTATIONAL RESULTS

5.1 Linear Kernal SVM:

Before performing the best feature selection

```
[LibSVM]Linear SVM with C=0.01: 30.79%
[LibSVM]Linear SVM with C=0.1: 29.91%
[LibSVM]Linear SVM with C=1: 39.20%
[LibSVM]Linear SVM with C=10: 48.01%
[LibSVM]Linear SVM with C=100: 48.76%
[LibSVM]Linear SVM with C=1000: 47.27%
```

After performing the best feature selection:

```
[LibSVM]Linear SVM with C=0.01: 84.52%
[LibSVM]Linear SVM with C=0.1: 84.50%
[LibSVM]Linear SVM with C=1: 82.18%
[LibSVM]Linear SVM with C=10: 54.07%
[LibSVM]Linear SVM with C=50: 56.50%
[LibSVM]Linear SVM with C=100: 61.47%
[LibSVM]Linear SVM with C=500: 74.03%
[LibSVM]Linear SVM with C=1000: 70.48%
```

The results show that Linear SVM with C = 0.01 after the feature selection gives best accuracy of 84.52%.

5.2 Radial basis function (RBF) kernel SVM:

Before performing the best feature selection

```
[LibSVM]Radial SVM with C=0.01: 71.56%
[LibSVM]Radial SVM with C=0.1: 72.46%
[LibSVM]Radial SVM with C=1: 71.27%
[LibSVM]Radial SVM with C=10: 69.49%
[LibSVM]Radial SVM with C=100: 67.70%
[LibSVM]Radial SVM with C=1000: 67.70%
```

After performing the best feature selection

```
[LibSVM]Radial SVM with C=0.01: 73.41%
[LibSVM]Radial SVM with C=0.1: 82.52%
[LibSVM]Radial SVM with C=1: 84.73%
[LibSVM]Radial SVM with C=10: 82.39%
[LibSVM]Radial SVM with C=100: 68.72%
[LibSVM]Radial SVM with C=1000: 70.28%
```

We can see C = 1 gives better accuracy of 84.73%.

```
[LibSVM]Radial SVM with gamma g=0.01: 84.29%
[LibSVM]Radial SVM with gamma g=0.1: 84.79%
[LibSVM]Radial SVM with gamma g=1: 84.73%
[LibSVM]Radial SVM with gamma g=10: 79.48%
```

Gamma = 0.1 and cost = 1 is giving the best result with accuracy 84.79%

5.3 Polynomial Kernel SVM:

Before performing the best feature selection

```
[LibSVM]Polynomial SVM with C=0.01: 85.16%
[LibSVM]Polynomial SVM with C=0.1: 85.73%
[LibSVM]Polynomial SVM with C=1: 85.94%
[LibSVM]Polynomial SVM with C=10: 55.06%
[LibSVM]Polynomial SVM with C=100: 32.92%
[LibSVM]Polynomial SVM with C=1000: 37.30%
```

After performing the best feature selection

```
[LibSVM]Polynomial SVM with C=0.01: 84.00%
[LibSVM]Polynomial SVM with C=0.1: 84.45%
[LibSVM]Polynomial SVM with C=1: 84.71%
[LibSVM]Polynomial SVM with C=10: 75.28%
[LibSVM]Polynomial SVM with C=100: 47.38%
[LibSVM]Polynomial SVM with C=1000: 60.88%
```

Trying different degree values.

```
[LibSVM]Polynomial SVM with degree d = 2: 84.45%
[LibSVM]Polynomial SVM with degree d = 3: 84.69%
[LibSVM]Polynomial SVM with degree d = 4: 85.03%
```

Though the highest accuracy decreased by a little after best subset selection, I still prefer using the subset as we have to train the data with higher degree polynomials and it may take a lot of computational time.

For polynomial SVM, the model with degree 4 and cost 0.1 is the model I selected. It has 85.03% test accuracy.

6. DISCUSSION

6.1 LINEAR KERNEL SVM:

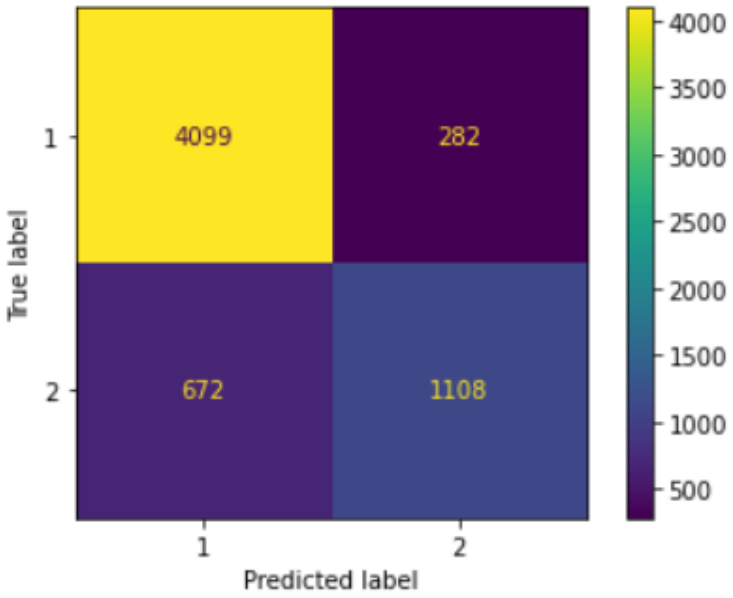


Fig. Confusion matrix of best linear kernel SVM model.

Total 5,207 people, 4099 owners and 1108 renters are correctly classified as the owners and renters respectively. 672 renters are misclassified as owners and 282 owners are misclassified as renters.

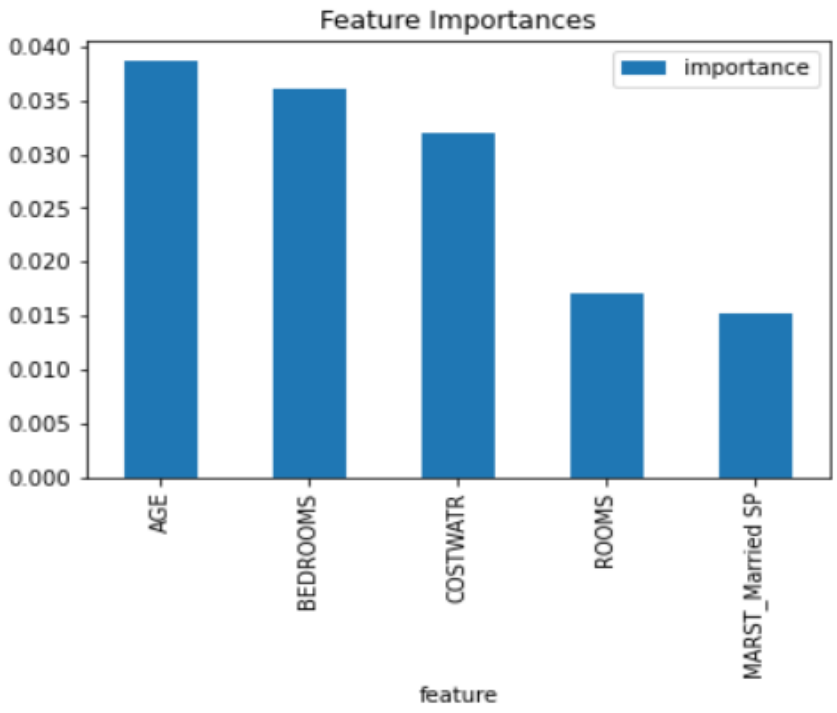
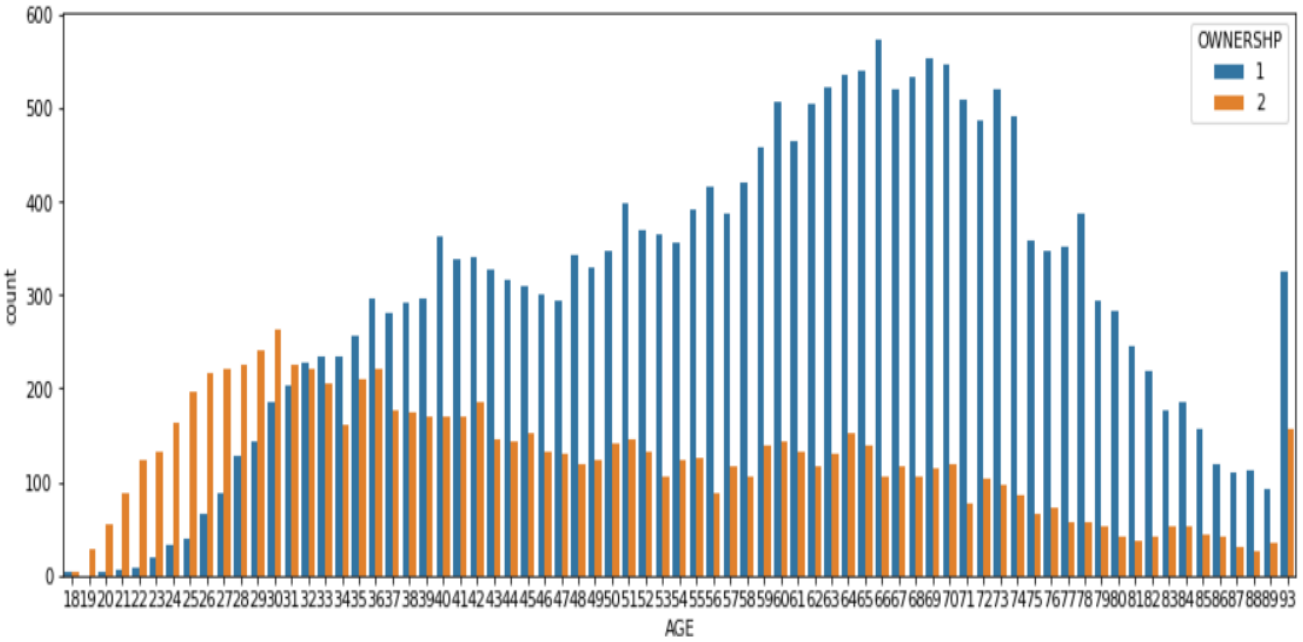
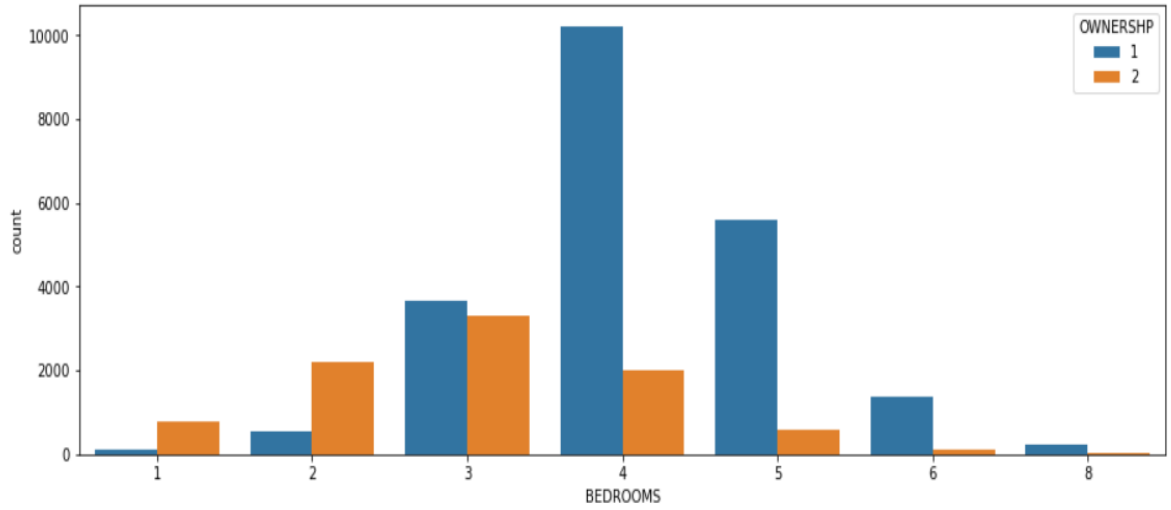


Fig. Feature Importances obtained by linear kernel svm to classify ownership status.

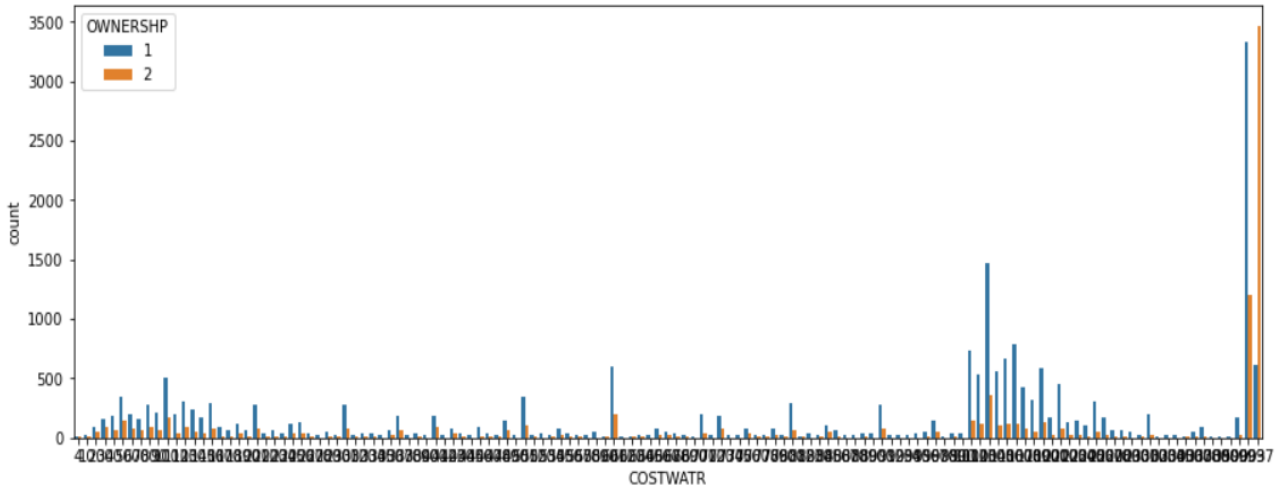
These are the 5 most important features to predict the ownership status. They are AGE of the person, number of BEDROOMS, COST OF WATER and number of ROOMS in the household and marital status where the person is married and has a spouse.



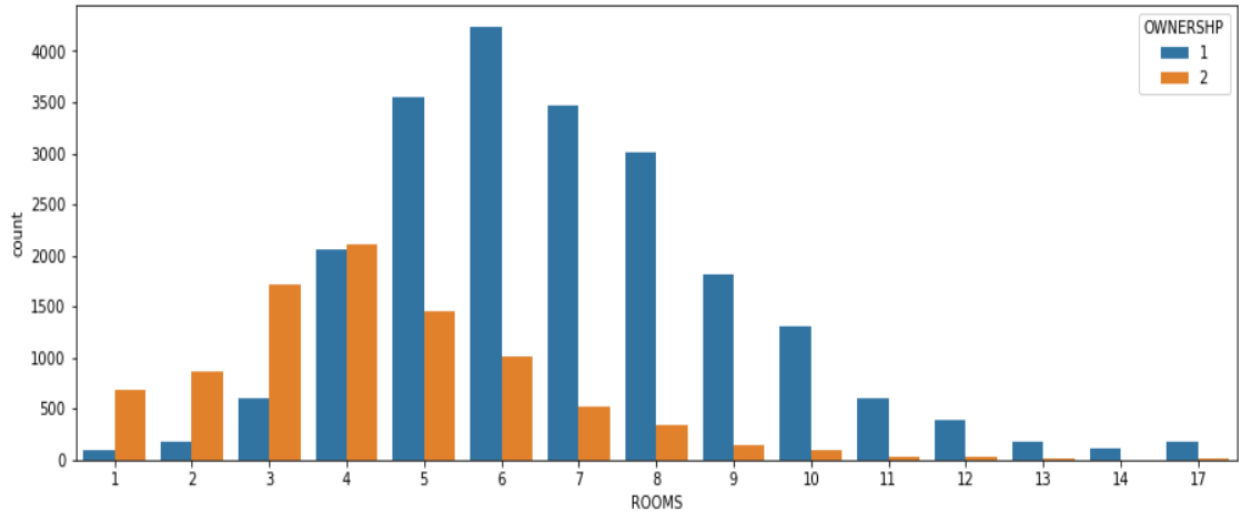
We can see that the most of the owners are increasing with AGE and then decreasing after a certain point of AGE say 75. It does make sense as the people tend to buy house after they settle in their life or to settle in their lives. After a certain age, they may give their ownership to their children.



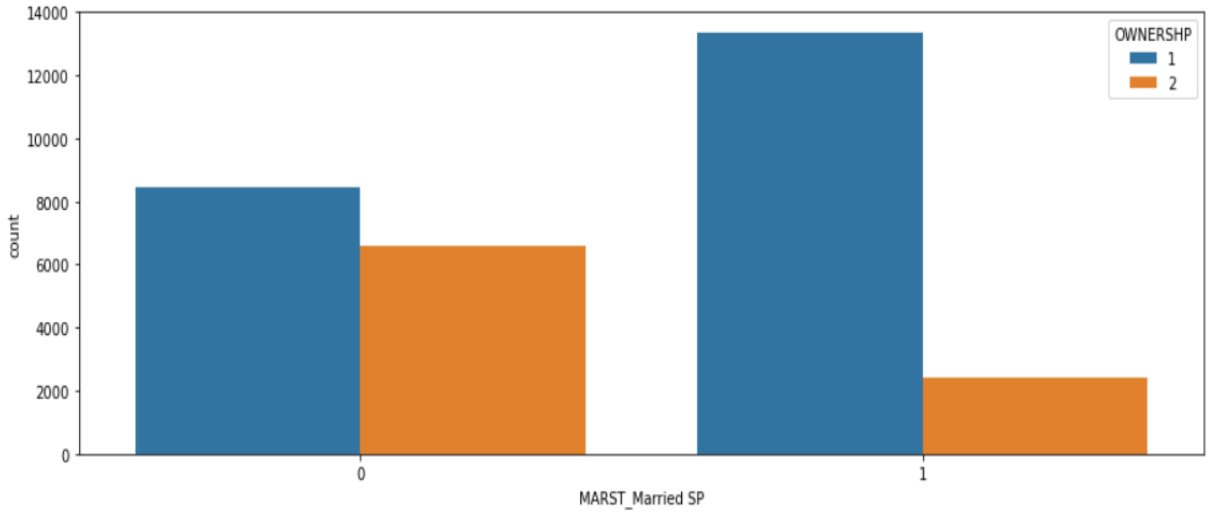
The households with 4 and 5 BEDROOMS have more positive ownership values. So, we can house with moderate number of BEDROOMS are most sold as people are willing to buy them more.



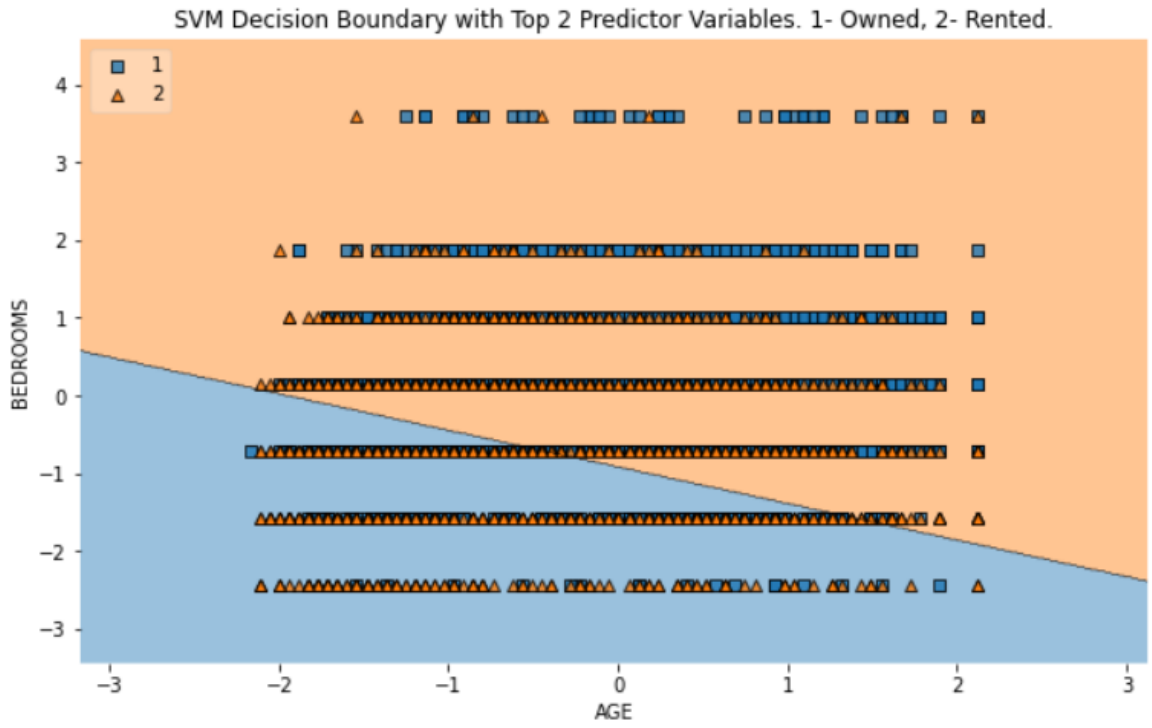
Even though the countplot doesn't look that good, we can see if the cost of water is extreme high, there are more renters. So, we can conclude people are not willing to buy those houses.



This one is similar to BEDROOMS scenario. The household with moderate ROOMS have more owners.



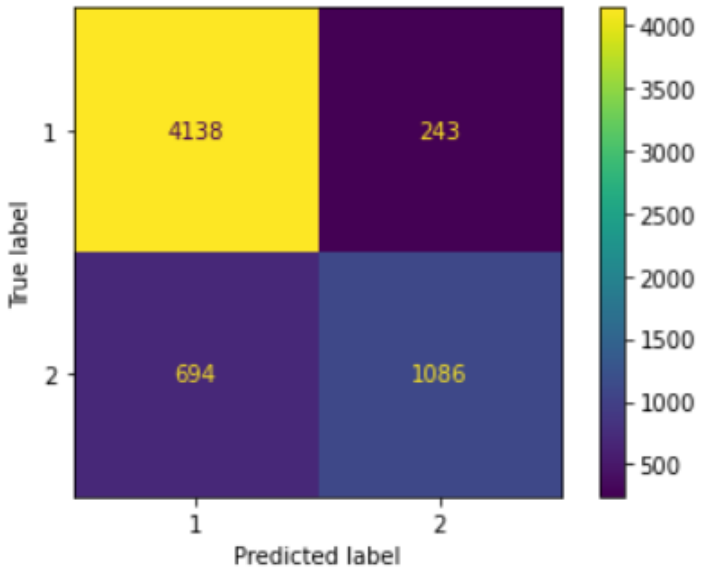
We can see people whose marital status is Married and Spouse is present are more owners. It makes sense that they are buying houses because they may want to settle at a place with the family.



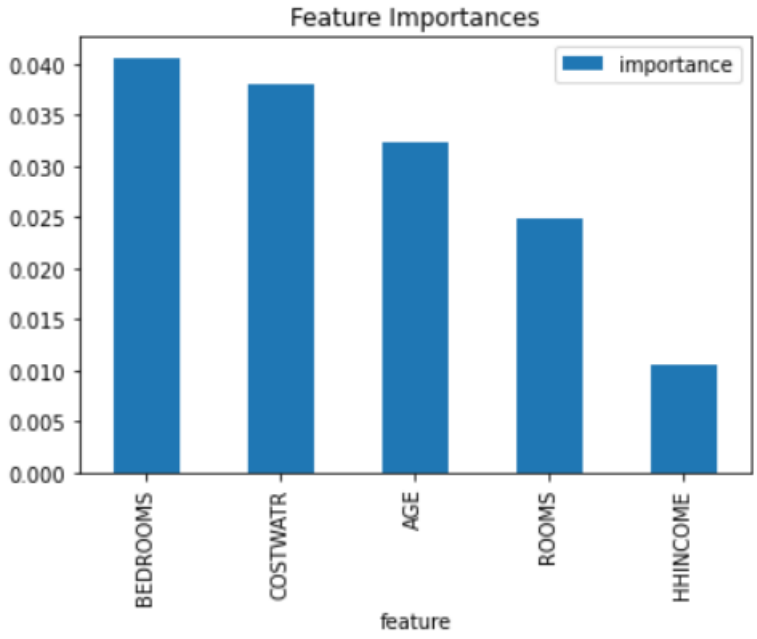
We can see a linear boundary as it is a linear kernel SVM. The lower region of feature space is assigned to OWNERS and upper region is assigned to RENTERS. We can see a lot of RENTERS are misclassified as OWNERS which is true even from the confusion matrix and few of the OWNERS are misclassified as RENTERS.

We cannot see many of the OWNER'S points as they may be in other view and may be the RENTERS values are overlapping with OWNERS in our view as there are many other variables that impact the OWNERSHIP.

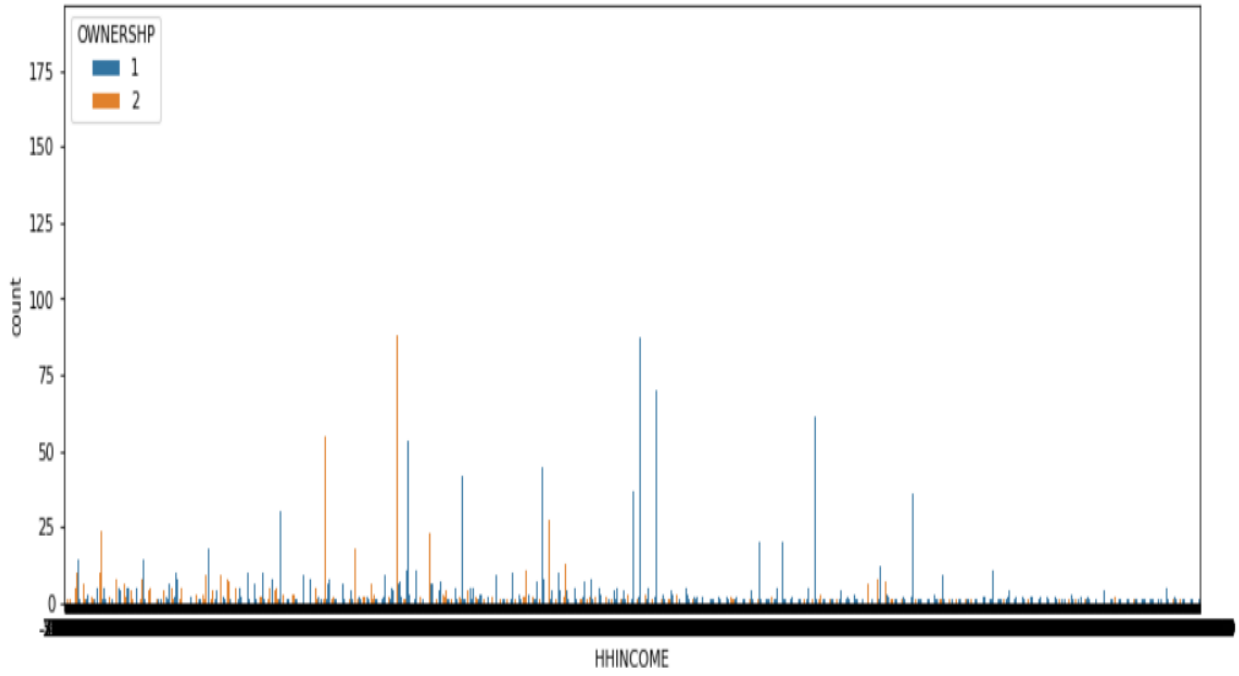
6.2 RADIAL KERNEL SVM:



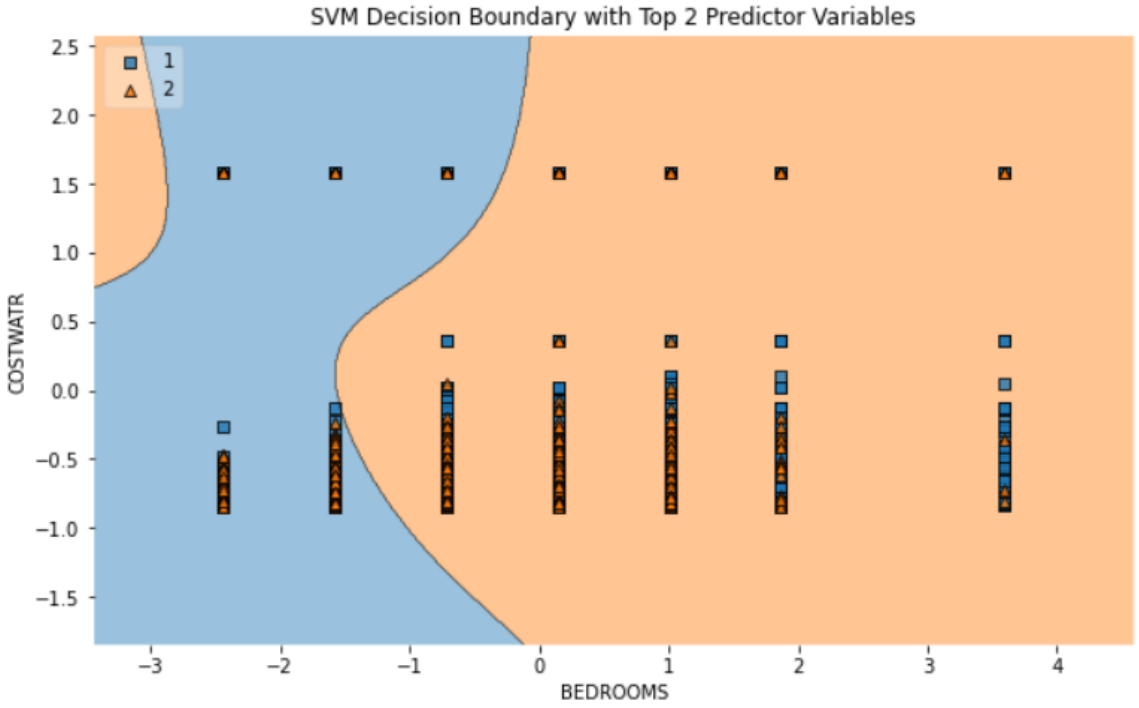
This is the confusion matrix for the Radial Kernal SVM. 694 values of RENTERS are misclassified as OWNERS and 243 values of OWNERS are misclassified as RENTERS.



All the important factors are already discussed in linear kernel SVM in 6.1. The only extra variable in top 5 is HHINCOME.

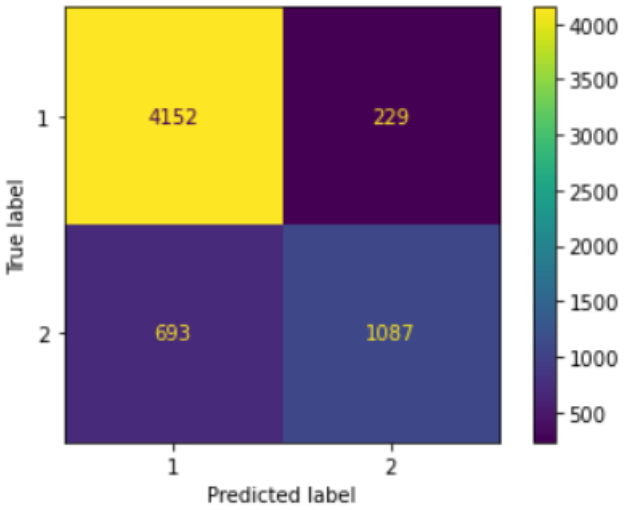


The graph looks kind of complicated, but we can see most of the RENTERS are from low-income households and most of the OWNERS are from high income households.

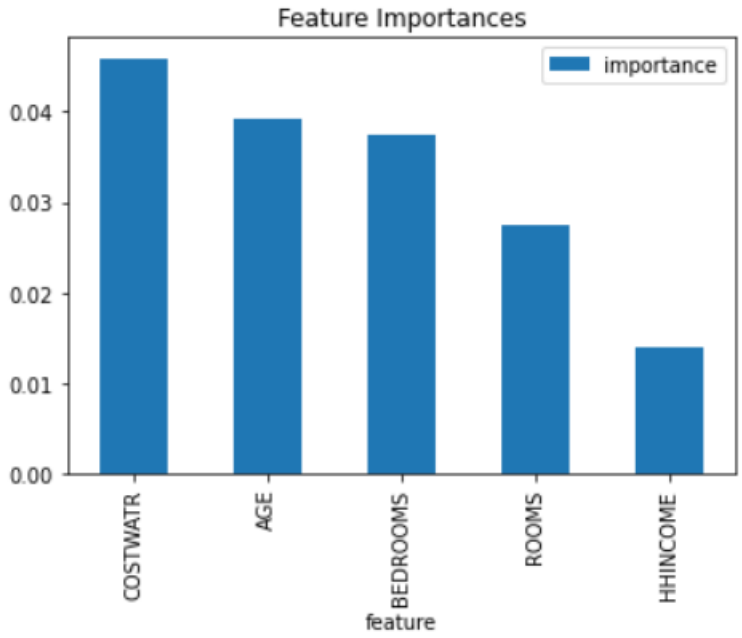


We can see the decision boundary is non-linear as we used radial kernel SVM. The blue region is the OWNERS region and ORANGE region is the RENTERS region. We can see few overlaps as there are still some misclassifications. The view of values may be better in other views as there are many variables. But we can see a clear hyperplane that is dividing the values in this view.

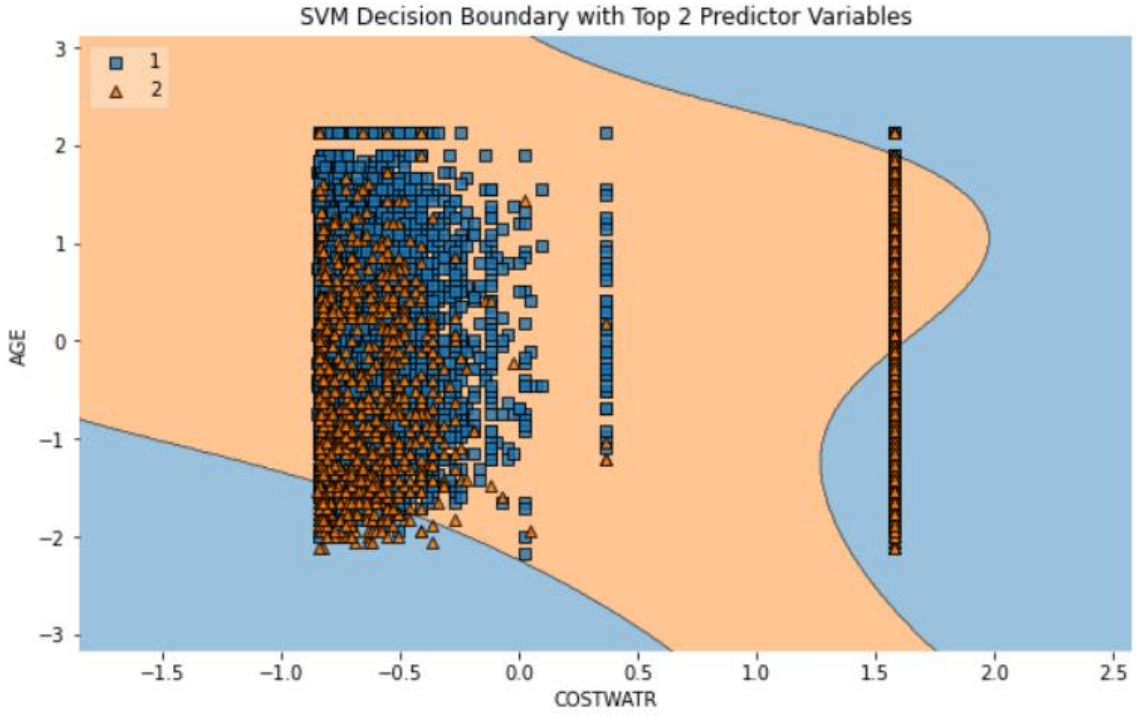
6.3 POLYNOMIAL KERNEL SVM:



This is the confusion matrix of polynomial kernel SVM. 693 of RENTERS are misclassified as OWNERS and 229 OWNERS are misclassified as RENTERS.



These are the important features according to polynomial kernel SVM. All of these variables are already discussed in linear and radial kernels in 6.2 and 6.3.



This is the decision boundary of a 4th degree polynomial. The blue region is OWNERS region and ORANGE belongs to RENTERS. We can still see a lot of misclassifications.

Based on all the results, I might consider using Radial as it has better results than linear and also less complex and less time taking compared to polynomial.

7. CONCLUSIONS

In this project, we aimed to predict home ownership based on demographic and socio-economic factors using Support Vector Machine (SVM) models. Our analysis revealed several important features that were strongly associated with home ownership.

Firstly, the age of the person was found to be the most important predictor of home ownership. This suggests that as people grow older, they are more likely to own their own homes. Additionally, the number of bedrooms and rooms in a household were also found to be important features for predicting home ownership. Households with four to five bedrooms and moderate numbers of rooms were more likely to have positive ownership values.

The cost of water was also found to be an important predictor of home ownership. Households with extreme high costs of water were more likely to have negative ownership values, indicating that people are less willing to buy houses with high water costs.

Even marital status was found to be an important predictor of home ownership. Specifically, people who were married and had a spouse were more likely to own their own homes.

Even household income is an important predictor of home ownership. People with low household income were more likely to have negative ownership values. and people with high income tend to own more houses.

Overall, these findings have important implications for policymakers, real estate agents, and homeownership advocacy groups. By understanding the key factors associated with home ownership, these groups can develop targeted interventions to help more people become homeowners, which can have positive impacts on social and economic outcomes.

8. BIBLIOGRAPHY

1. Ruggles, S., Flood, S., Sobek, M., Brockman, D., Cooper, G., Richards, S., & Schouweiler, M. (2023). IPUMS USA: Version 13.0 [dataset]. Minneapolis, MN: IPUMS. https://doi.org/10.18128/D010.V13.0

2. scikit-learn: Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, É. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825-2830. https://scikit-learn.org/stable/about.html#citing-scikit-learn

3. "Support Vector Machines" scikit-learn documentation. Available online: https://scikit-learn.org/stable/modules/svm.html

9.APPENDIX

In [1]:
```
# Import libraries
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt

from sklearn.svm import SVC
from sklearn.metrics import roc_curve
from mlxtend.plotting import plot_decision_regions
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score
from sklearn.feature_selection import RFE
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.inspection import permutation_importance
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.feature_selection import RFECV, SelectKBest
from sklearn.pipeline import Pipeline
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
df = pd.read_csv('Housing.csv')
df.shape
```

Out[2]:
```
(75388, 23)
```

In [3]:
```
df.columns
```

Out[3]:
```
Index(['SERIAL', 'DENSITY', 'OWNERSHP', 'OWNERSHPD', 'COSTELEC', 'COSTGAS',
       'COSTWATR', 'COSTFUEL', 'HHINCOME', 'ROOMS', 'BUILTYR2', 'BEDROOMS',
       'VEHICLES', 'NFAMS', 'NCOUPLES', 'PERNUM', 'PERWT', 'AGE', 'MARST',
       'BIRTHYR', 'EDUC', 'EDUCD', 'INCTOT'],
      dtype='object')
```

DATA SUBSETTING

In [4]:
```
# Consider only elder person rows for every household
data = df.sort_values(['SERIAL', 'AGE'], ascending=[True, False]).drop_duplicates('SER
data.shape
```

Out[4]:
```
(30802, 23)
```

In [5]:
```
#Dropping the columns that seems not necessary
data = data.drop(['SERIAL', 'OWNERSHPD', 'PERNUM', 'PERWT', 'BIRTHYR', 'EDUC', 'EDUCD'
```

In [6]:
```
# checking for any NULL Values
data.isnull().any(axis=1).sum()
```

Out[6]:
```
0
```

In [7]:
```
data.columns
```

Out[7]:
```
Index(['DENSITY', 'OWNERSHP', 'COSTELEC', 'COSTGAS', 'COSTWATR', 'COSTFUEL',
       'HHINCOME', 'ROOMS', 'BUILTYR2', 'BEDROOMS', 'VEHICLES', 'NFAMS',
       'NCOUPLES', 'AGE', 'MARST', 'INCTOT'],
      dtype='object')
```

In [8]:
```
data['MARST'] = data['MARST'].replace({1: 'Married SP', 2: 'Married SA', 3: 'Separated
data = pd.get_dummies(data, columns=['MARST'])
data.columns
```

Out[8]:
```
Index(['DENSITY', 'OWNERSHP', 'COSTELEC', 'COSTGAS', 'COSTWATR', 'COSTFUEL',
       'HHINCOME', 'ROOMS', 'BUILTYR2', 'BEDROOMS', 'VEHICLES', 'NFAMS',
       'NCOUPLES', 'AGE', 'INCTOT', 'MARST_Divorced', 'MARST_Married SA',
       'MARST_Married SP', 'MARST_Never married', 'MARST_Separated',
       'MARST_Widowed'],
      dtype='object')
```

In [9]:
```
# Split data into X and y
X = data.drop('OWNERSHP', axis=1)
y = data['OWNERSHP']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=

# Scale the training and testing data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

LINEAR

In [10]:
```
C_values = [0.01, 0.1, 1, 10, 100, 1000]
for C in C_values:
    linear_svc = SVC(kernel='linear', C=C, cache_size=1000, verbose = True, max_iter =

    # Fit the model to the training data
    linear_svc.fit(X_train_scaled, y_train)

    # Evaluate the model on the testing data and print the accuracy score
    accuracy = linear_svc.score(X_test_scaled, y_test)
    print("Linear SVM with C=%s: %.2f%%" % (C, accuracy * 100))
```

```
[LibSVM]Linear SVM with C=0.01: 30.79%
[LibSVM]Linear SVM with C=0.1: 29.91%
[LibSVM]Linear SVM with C=1: 39.20%
[LibSVM]Linear SVM with C=10: 48.01%
[LibSVM]Linear SVM with C=100: 48.76%
[LibSVM]Linear SVM with C=1000: 47.27%
```

FEATURE SELECTION

In [11]:
```
# Perform feature selection
k = 10  # number of top features to select
selector = SelectKBest(f_classif, k=k)
selector.fit(X_train_scaled, y_train)
```

```
X_train_new = selector.transform(X_train_scaled)
X_test_new = selector.transform(X_test_scaled)
```

In [12]:
```
C_values = [0.01, 0.1, 1, 10, 50, 100, 500, 1000]

for C in C_values:
    linear_svc = SVC(kernel='linear', C=C, cache_size=1000, verbose = True, max_iter =

    # Fit the model to the training data
    linear_svc.fit(X_train_new, y_train)

    # Evaluate the model on the testing data and print the accuracy score
    accuracy = linear_svc.score(X_test_new, y_test)
    print("Linear SVM with C=%s: %.2f%%" % (C, accuracy * 100))
```

```
[LibSVM]Linear SVM with C=0.01: 84.52%
[LibSVM]Linear SVM with C=0.1: 84.50%
[LibSVM]Linear SVM with C=1: 82.18%
[LibSVM]Linear SVM with C=10: 54.07%
[LibSVM]Linear SVM with C=50: 56.50%
[LibSVM]Linear SVM with C=100: 61.47%
[LibSVM]Linear SVM with C=500: 74.03%
[LibSVM]Linear SVM with C=1000: 70.48%
```

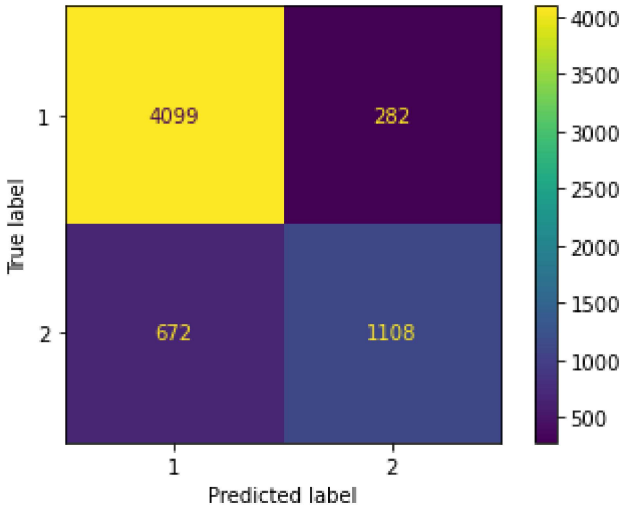
C = 0.01 is the best model with accuracy 84.52%

In [13]:
```
linear_svc = SVC(kernel='linear', C=0.01, cache_size=1000, verbose = True, max_iter =

# Fit the model to the training data
linear_svc.fit(X_train_new, y_train)

# Evaluate the model on the testing data and print the accuracy score
accuracy = linear_svc.score(X_test_new, y_test)
print("Linear SVM with C=0.01: %.2f%%" % (accuracy * 100))
```

```
[LibSVM]Linear SVM with C=0.01: 84.52%
```

In [14]:
```
# plot the confusion matrix
y_pred = linear_svc.predict(X_test_new)
disp = ConfusionMatrixDisplay.from_predictions(y_test, y_pred)
disp
```

Out[14]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2416108f700>
```



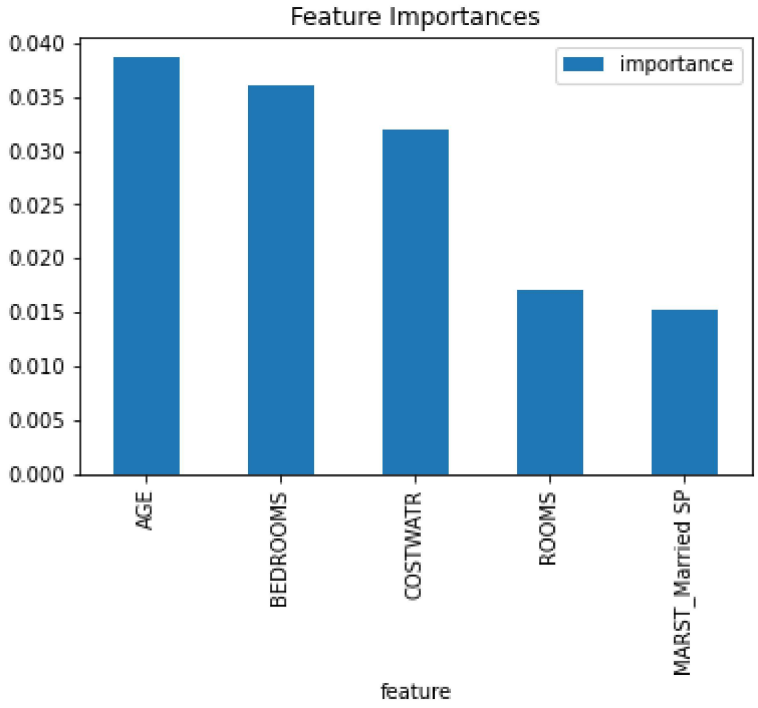
In [16]:
```
# Calculate feature importances using permutation importance
result = permutation_importance(linear_svc, X_test_new, y_test, n_repeats=5, random_st
importance_df = pd.DataFrame({'feature': X.columns[selector.get_support()], 'importanc
importance_df = importance_df.sort_values(by='importance', ascending=False).reset_inde
```

In [17]:
```
importance_df
```

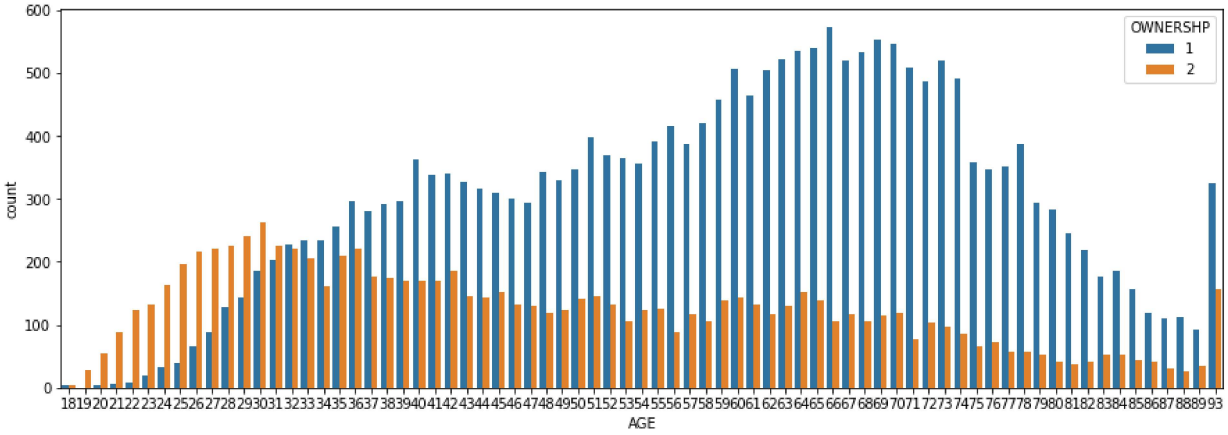
Out[17]:

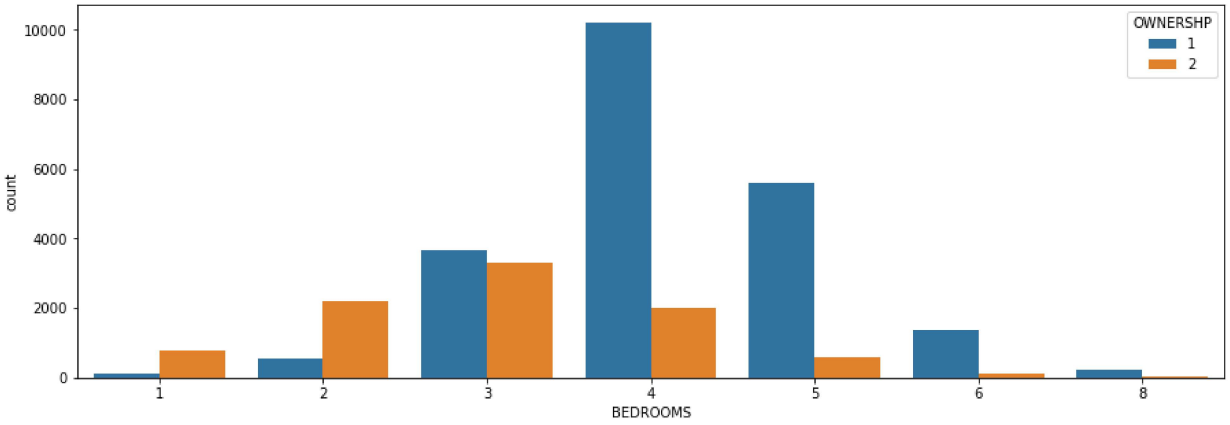
	feature	importance
0	AGE	0.038598
1	BEDROOMS	0.036066
2	COSTWATR	0.031943
3	ROOMS	0.017108
4	MARST_Married SP	0.015290
5	DENSITY	0.013245
6	HHINCOME	0.012206
7	COSTGAS	0.008148
8	MARST_Never married	0.000519
9	NCOUPLES	0.000390

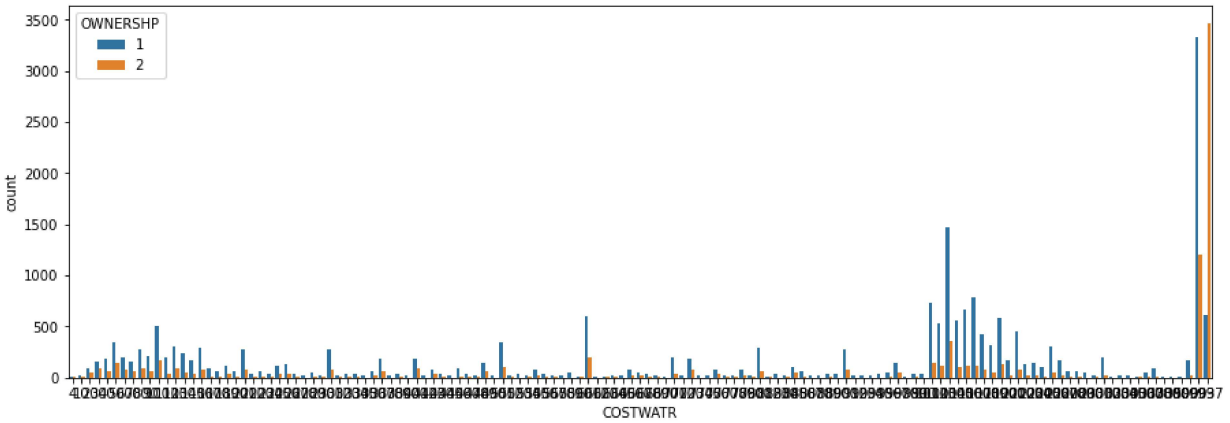
In [18]:
```
importance_df.iloc[:5].plot(kind='bar', x='feature', y='importance')
plt.title('Feature Importances')
plt.show()
```

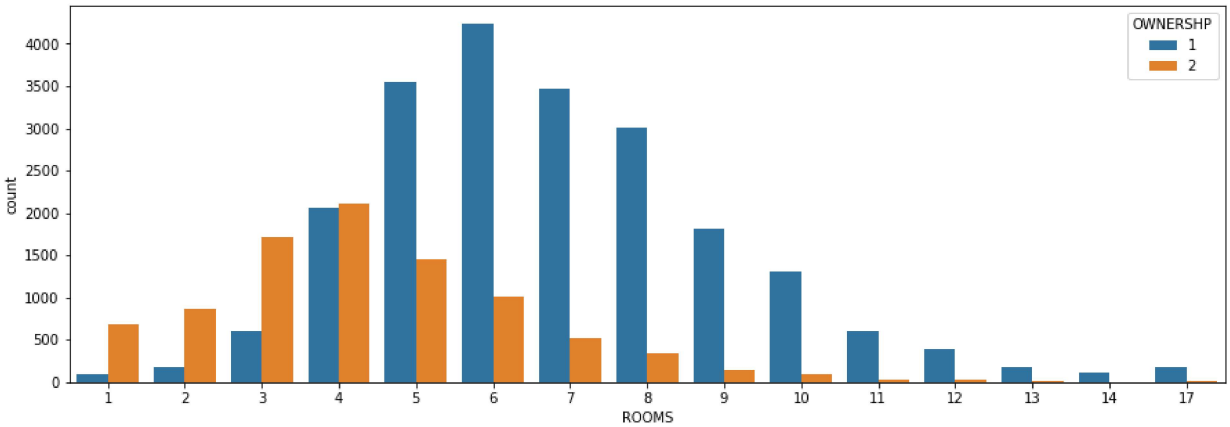


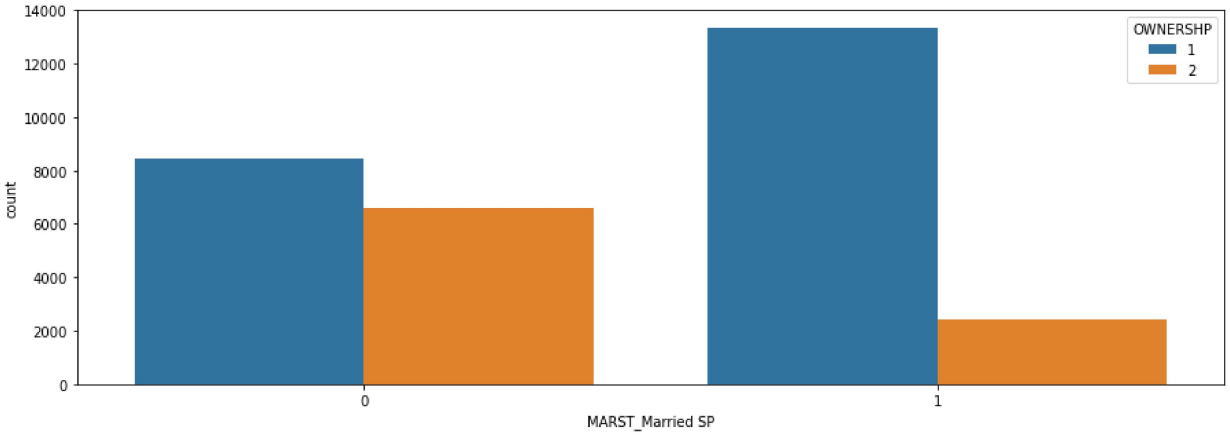
In [19]:
```
import seaborn as sns

# Get the top 5 important variables
top_5_vars = importance_df.iloc[:5]['feature'].tolist()

# Plot countplots for each of the top 5 variables
for var in top_5_vars:
    plt.figure(figsize=(15, 5))
    sns.countplot(x=var, hue='OWNERSHP', data=data)
    plt.show()
```









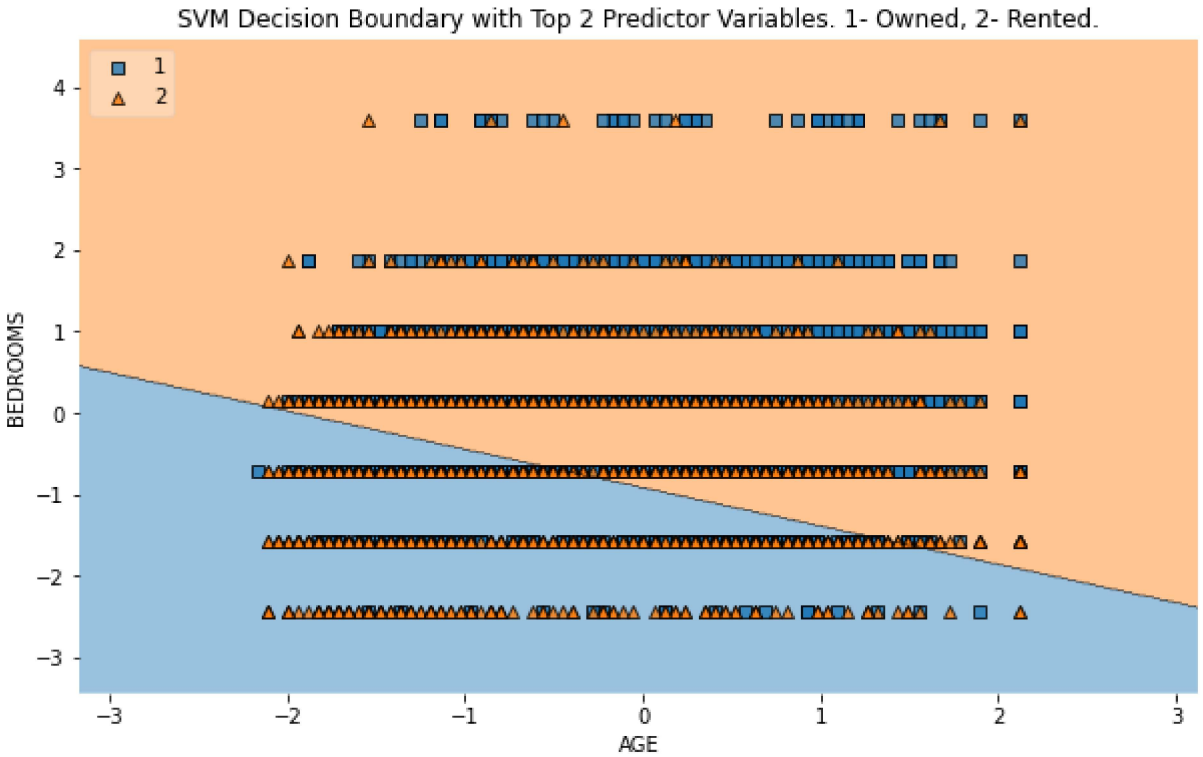


In [20]:
```
# Get the top 2 predictor variables
top_predictors = importance_df.head(2)['feature'].values
print("Top 2 predictwor variables:", top_predictors)
```

```
Top 2 predictwor variables: ['AGE' 'BEDROOMS']
```

In [21]:
```
# Filter the dataset to keep only the top 2 predictor variables
X_top_predictors = X[top_predictors]

# Split the data into training and testing sets
X_train_top, X_test_top, y_train_top, y_test_top = train_test_split(X_top_predictors,

# Scale the training and testing data
scaler_top = StandardScaler()
X_train_top_scaled = scaler_top.fit_transform(X_train_top)
X_test_top_scaled = scaler_top.transform(X_test_top)

# Train the best model with the top 2 predictor variables
linear_svc_top = SVC(kernel='linear', C=0.01, cache_size=1000, verbose=True, max_iter=
linear_svc_top.fit(X_train_top_scaled, y_train_top)

# Plot the decision boundary
plt.figure(figsize=(10, 6))
plot_decision_regions(X_test_top_scaled, y_test_top.to_numpy(), clf=linear_svc_top, le
plt.xlabel(top_predictors[0])
plt.ylabel(top_predictors[1])
plt.title('SVM Decision Boundary with Top 2 Predictor Variables. 1- Owned, 2- Rented.'
plt.show()
```

```
[LibSVM]
```



RADIAL

In [22]:
```
C_values = [0.01, 0.1, 1, 10, 100, 1000]
for C in C_values:
    radial_svc = SVC(kernel='rbf', gamma=1, C=C, cache_size=1000, verbose = True, max_

    # Fit the model to the training data
```

```
    radial_svc.fit(X_train_scaled, y_train)

    # Evaluate the model on the testing data and print the accuracy score
    accuracy = radial_svc.score(X_test_scaled, y_test)
    print("Radial SVM with C=%s: %.2f%%" % (C, accuracy * 100))
```

```
[LibSVM]Radial SVM with C=0.01: 71.56%
[LibSVM]Radial SVM with C=0.1: 72.46%
[LibSVM]Radial SVM with C=1: 71.27%
[LibSVM]Radial SVM with C=10: 69.49%
[LibSVM]Radial SVM with C=100: 67.70%
[LibSVM]Radial SVM with C=1000: 67.70%
```

FEATURE SELECTION

In [23]:
```
# Perform feature selection
k = 10  # number of top features to select
selector = SelectKBest(f_classif, k=k)
selector.fit(X_train_scaled, y_train)
X_train_new = selector.transform(X_train_scaled)
X_test_new = selector.transform(X_test_scaled)
```

In [28]:
```
C_values = [0.01, 0.1, 1, 10, 100, 1000]

for C in C_values:
    radial_svc = SVC(kernel='rbf',gamma=1, C=C, cache_size=1000, verbose = True, max_i

    # Fit the model to the training data
    radial_svc.fit(X_train_new, y_train)

    # Evaluate the model on the testing data and print the accuracy score
    accuracy = radial_svc.score(X_test_new, y_test)
    print("Radial SVM with C=%s: %.2f%%" % (C, accuracy * 100))
```

```
[LibSVM]Radial SVM with C=0.01: 73.41%
[LibSVM]Radial SVM with C=0.1: 82.52%
[LibSVM]Radial SVM with C=1: 84.73%
[LibSVM]Radial SVM with C=10: 82.39%
[LibSVM]Radial SVM with C=100: 68.72%
[LibSVM]Radial SVM with C=1000: 70.28%
```

In [29]:
```
g_values = [0.01, 0.1, 1, 10]

for g in g_values:
    radial_svc = SVC(kernel='rbf',gamma=g, C=1, cache_size=1000, verbose = True, max_i

    # Fit the model to the training data
    radial_svc.fit(X_train_new, y_train)

    # Evaluate the model on the testing data and print the accuracy score
    accuracy = radial_svc.score(X_test_new, y_test)
    print("Radial SVM with gamma g=%s: %.2f%%" % (g, accuracy * 100))
```

```
[LibSVM]Radial SVM with gamma g=0.01: 84.29%
[LibSVM]Radial SVM with gamma g=0.1: 84.79%
[LibSVM]Radial SVM with gamma g=1: 84.73%
[LibSVM]Radial SVM with gamma g=10: 79.48%
```

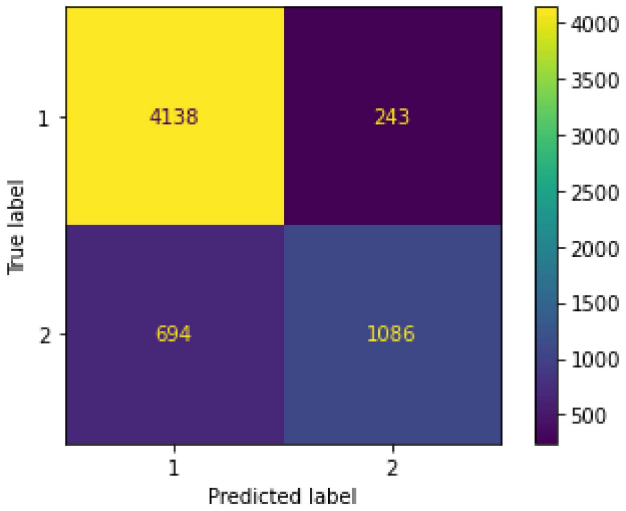
Gamma = 0.1 and cost = 1 is giving the best result with accuracy 84.79%

In [30]:
```
radial_svc = SVC(kernel='rbf',gamma=0.1, C=1, cache_size=1000, verbose = True, max_ite

# Fit the model to the training data
radial_svc.fit(X_train_new, y_train)

# Evaluate the model on the testing data and print the accuracy score
accuracy = radial_svc.score(X_test_new, y_test)
print("Radial SVM with gamma g= 0.1 and c = 1: %.2f%%" % (accuracy * 100))
```

```
[LibSVM]Radial SVM with gamma g= 0.1 and c = 1: 84.79%
```

In [31]:
```
# plot the confusion matrix
y_pred = radial_svc.predict(X_test_new)
disp = ConfusionMatrixDisplay.from_predictions(y_test, y_pred)
disp
```

Out[31]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2412d8ba070>
```



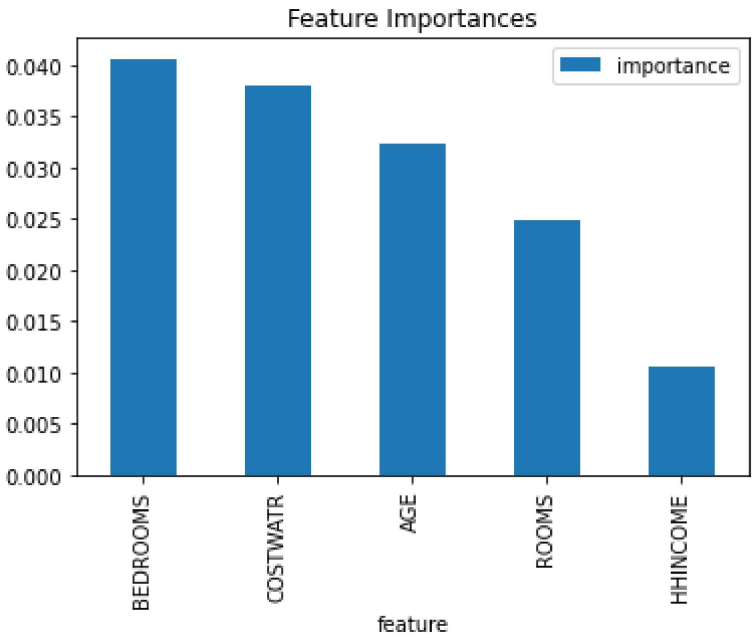
In [32]:
```
# Calculate feature importances using permutation importance
result = permutation_importance(radial_svc, X_test_new, y_test, n_repeats=5, random_st
importance_df = pd.DataFrame({'feature': X.columns[selector.get_support()], 'importanc
importance_df = importance_df.sort_values(by='importance', ascending=False).reset_inde
```

In [33]:
```
importance_df
```

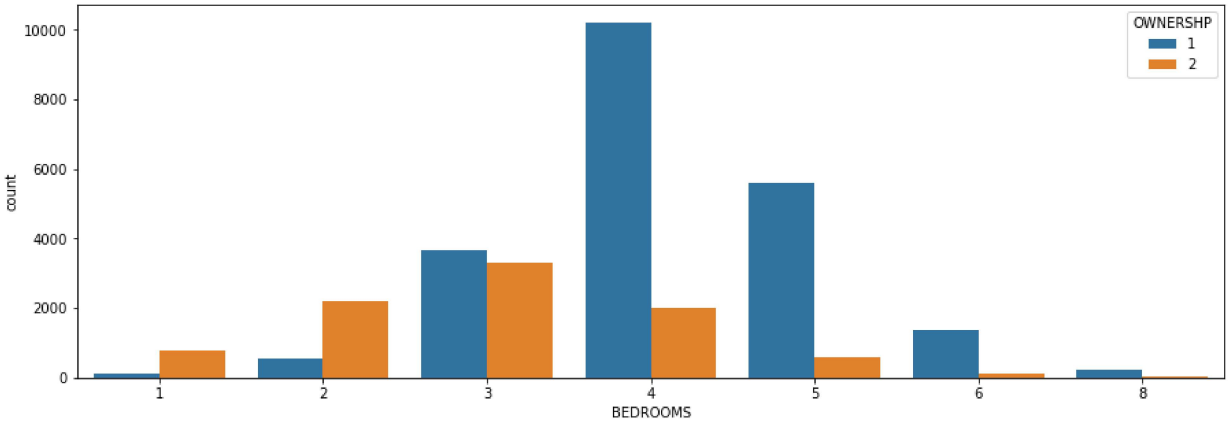
Out[33]:

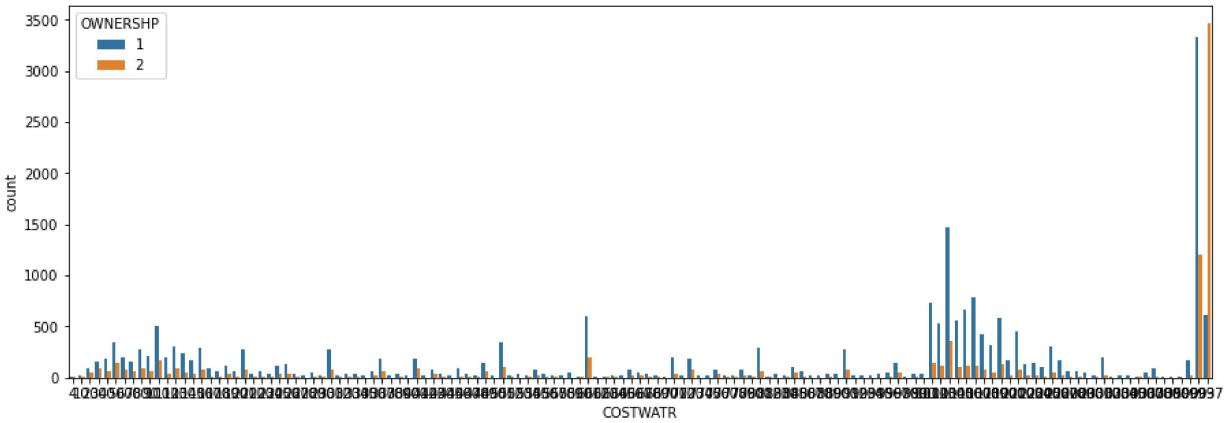
	feature	importance
0	BEDROOMS	0.040675
1	COSTWATR	0.038111
2	AGE	0.032397
3	ROOMS	0.024866
4	HHINCOME	0.010485
5	DENSITY	0.009998
6	COSTGAS	0.005551
7	MARST_Married SP	0.004902
8	NCOUPLES	0.001428
9	MARST_Never married	0.001006

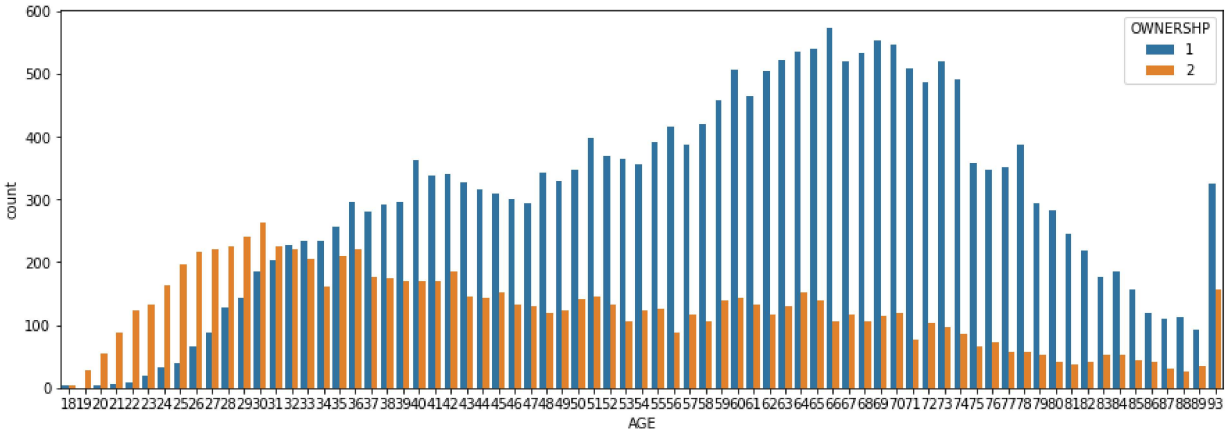
In [34]:
```
importance_df.iloc[:5].plot(kind='bar', x='feature', y='importance')
plt.title('Feature Importances')
plt.show()
```

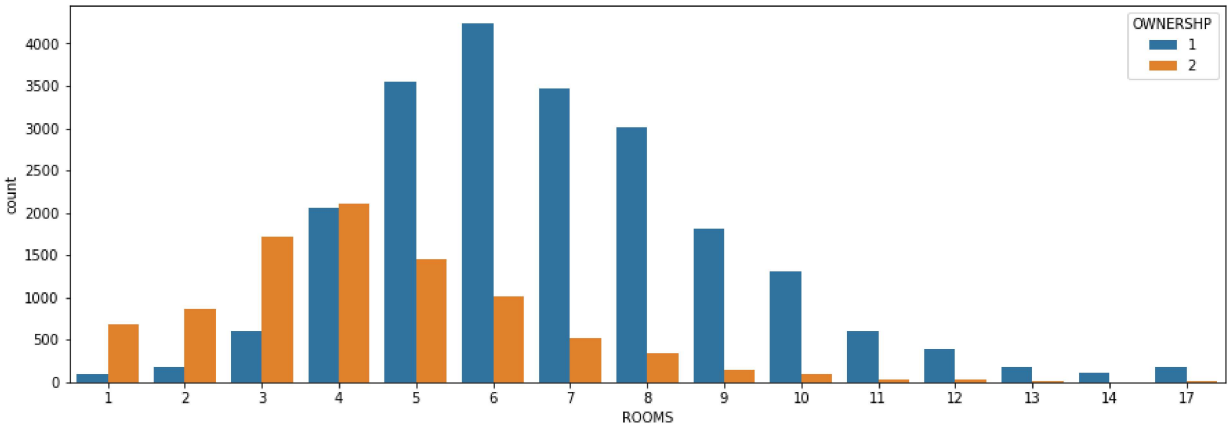


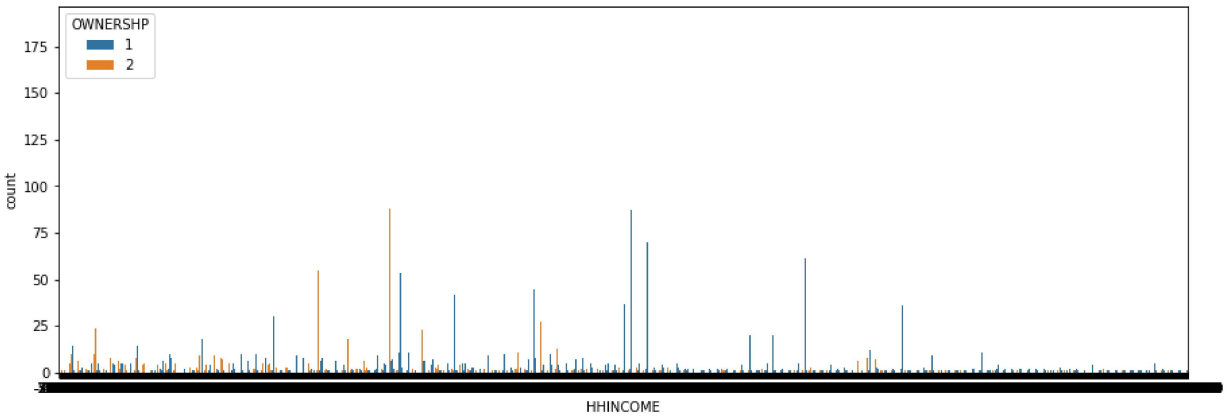
In [35]:
```
import seaborn as sns

# Get the top 5 important variables
top_5_vars = importance_df.iloc[:5]['feature'].tolist()

# Plot countplots for each of the top 5 variables
for var in top_5_vars:
    plt.figure(figsize=(15, 5))
    sns.countplot(x=var, hue='OWNERSHP', data=data)
    plt.show()
```


OWNERSHP
1
2
count
BEDROOMS


OWNERSHP
1
2
count
COSTWATR


OWNERSHP
1
2
count
AGE


OWNERSHP
1
2
count
ROOMS

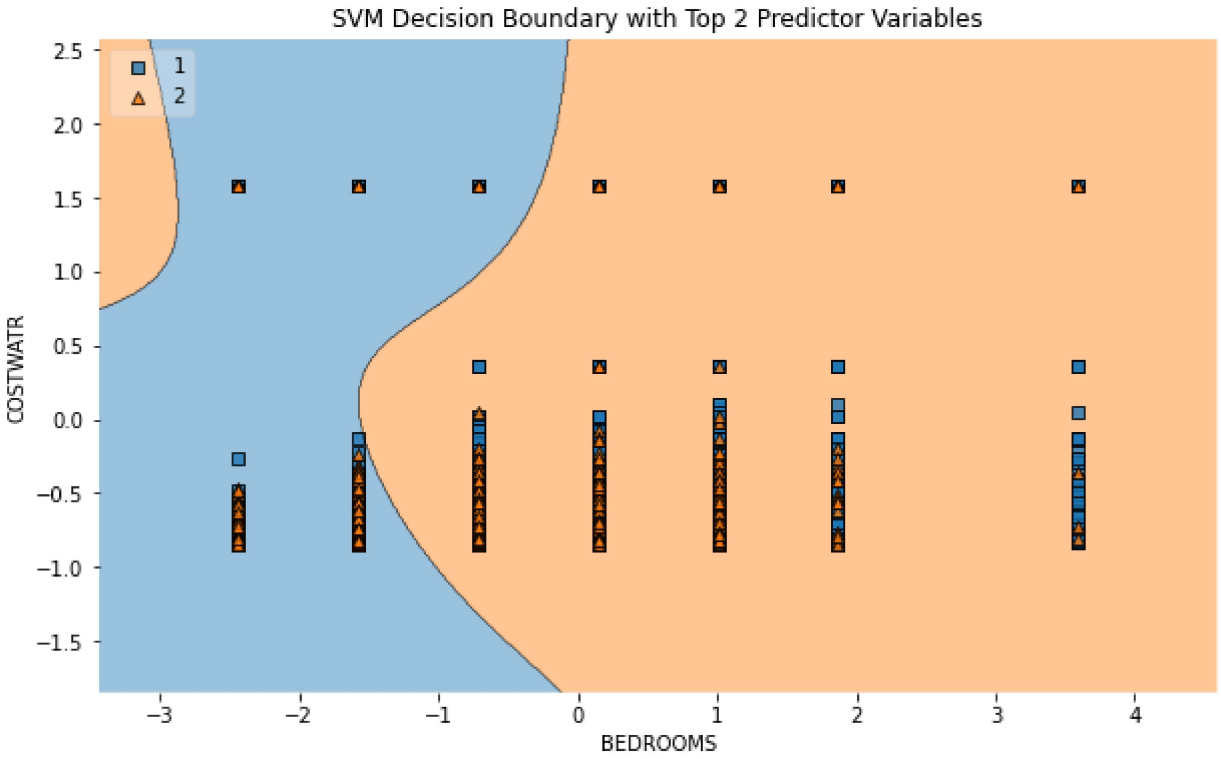


In [36]:
```
# Get the top 2 predictor variables
top_predictors = importance_df.head(2)['feature'].values
print("Top 2 predictor variables:", top_predictors)
```

```
Top 2 predictor variables: ['BEDROOMS' 'COSTWATR']
```

In [37]:
```
# Filter the dataset to keep only the top 2 predictor variables
X_top_predictors = X[top_predictors]

# Split the data into training and testing sets
X_train_top, X_test_top, y_train_top, y_test_top = train_test_split(X_top_predictors,

# Scale the training and testing data
scaler_top = StandardScaler()
X_train_top_scaled = scaler_top.fit_transform(X_train_top)
X_test_top_scaled = scaler_top.transform(X_test_top)

# Train the best model with the top 2 predictor variables
radial_svc_top = SVC(kernel='rbf', C=1, gamma = 0.1, cache_size=1000, verbose=True, ma
radial_svc_top.fit(X_train_top_scaled, y_train_top)

# Plot the decision boundary
plt.figure(figsize=(10, 6))
plot_decision_regions(X_test_top_scaled, y_test_top.to_numpy(), clf=radial_svc_top, le
plt.xlabel(top_predictors[0])
plt.ylabel(top_predictors[1])
plt.title('SVM Decision Boundary with Top 2 Predictor Variables')
plt.show()
```

```
[LibSVM]
```



POLYNOMIAL

In [38]:
```
C_values = [0.01, 0.1, 1, 10, 100, 1000]
for C in C_values:
    poly_svc = SVC(kernel='poly', degree=2, coef0=1, C=C, gamma='scale', cache_size=1(

    # Fit the model to the training data
    poly_svc.fit(X_train_scaled, y_train)

    # Evaluate the model on the testing data and print the accuracy score
    accuracy = poly_svc.score(X_test_scaled, y_test)
    print("Polynomial SVM with C=%s: %.2f%%" % (C, accuracy * 100))
```

```
[LibSVM]Polynomial SVM with C=0.01: 85.16%
[LibSVM]Polynomial SVM with C=0.1: 85.73%
[LibSVM]Polynomial SVM with C=1: 85.94%
[LibSVM]Polynomial SVM with C=10: 55.06%
[LibSVM]Polynomial SVM with C=100: 32.92%
[LibSVM]Polynomial SVM with C=1000: 37.30%
```

FEATURE SELECTION

In [39]:
```
# Perform feature selection
k = 10  # number of top features to select
selector = SelectKBest(f_classif, k=k)
selector.fit(X_train_scaled, y_train)
X_train_new = selector.transform(X_train_scaled)
X_test_new = selector.transform(X_test_scaled)
```

In [40]:
```
C_values = [0.01, 0.1, 1, 10, 100, 1000]

for C in C_values:
```

```
    poly_svc = SVC(kernel='poly', degree=2, coef0=1, C=C, gamma='scale', cache_size=1(

    # Fit the model to the training data
    poly_svc.fit(X_train_new, y_train)

    # Evaluate the model on the testing data and print the accuracy score
    accuracy = poly_svc.score(X_test_new, y_test)
    print("Polynomial SVM with C=%s: %.2f%%" % (C, accuracy * 100))
```

```
[LibSVM]Polynomial SVM with C=0.01: 84.00%
[LibSVM]Polynomial SVM with C=0.1: 84.45%
[LibSVM]Polynomial SVM with C=1: 84.71%
[LibSVM]Polynomial SVM with C=10: 75.28%
[LibSVM]Polynomial SVM with C=100: 47.38%
[LibSVM]Polynomial SVM with C=1000: 60.88%
```

In [42]:
```
degree_values = [2,3,4]

for d in degree_values:
    poly_svc = SVC(kernel='poly', degree=d, coef0=1, C= 1 , gamma='scale', cache_size=

    # Fit the model to the training data
    poly_svc.fit(X_train_new, y_train)

    # Evaluate the model on the testing data and print the accuracy score
    accuracy = poly_svc.score(X_test_new, y_test)
    print("Polynomial SVM with degree d = %s: %.2f%%" % (d, accuracy * 100))
```

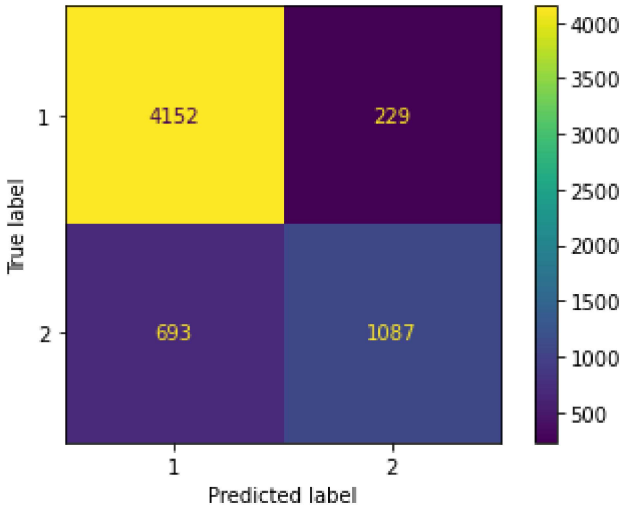
```
[LibSVM]Polynomial SVM with degree d = 2: 84.45%
[LibSVM]Polynomial SVM with degree d = 3: 84.69%
[LibSVM]Polynomial SVM with degree d = 4: 85.03%
```

In [43]:
```
poly_svc = SVC(kernel='poly', degree=4 , coef0=1, C=0.1 , gamma='scale', cache_size=1(

# Fit the model to the training data
poly_svc.fit(X_train_new, y_train)

# Evaluate the model on the testing data and print the accuracy score
accuracy = poly_svc.score(X_test_new, y_test)
print("Polynomial SVM with degree d = 4 and c = 0.1: %.2f%%" % (accuracy * 100))
```

```
[LibSVM]Polynomial SVM with degree d = 4 and c = 0.1: 85.03%
```

In [44]:
```
# plot the confusion matrix
y_pred = poly_svc.predict(X_test_new)
disp = ConfusionMatrixDisplay.from_predictions(y_test, y_pred)
disp
```

Out[44]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2412d1933d0>
```



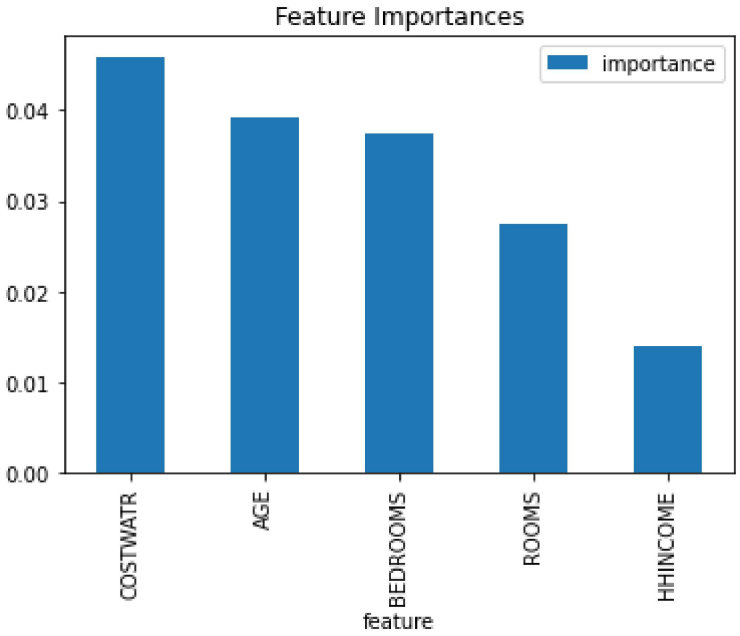
In [45]:
```
# Calculate feature importances using permutation importance
result = permutation_importance(poly_svc, X_test_new, y_test, n_repeats=10, random_sta
importance_df = pd.DataFrame({'feature': X.columns[selector.get_support()], 'importanc
importance_df = importance_df.sort_values(by='importance', ascending=False).reset_inde
```

In [46]:
```
importance_df
```

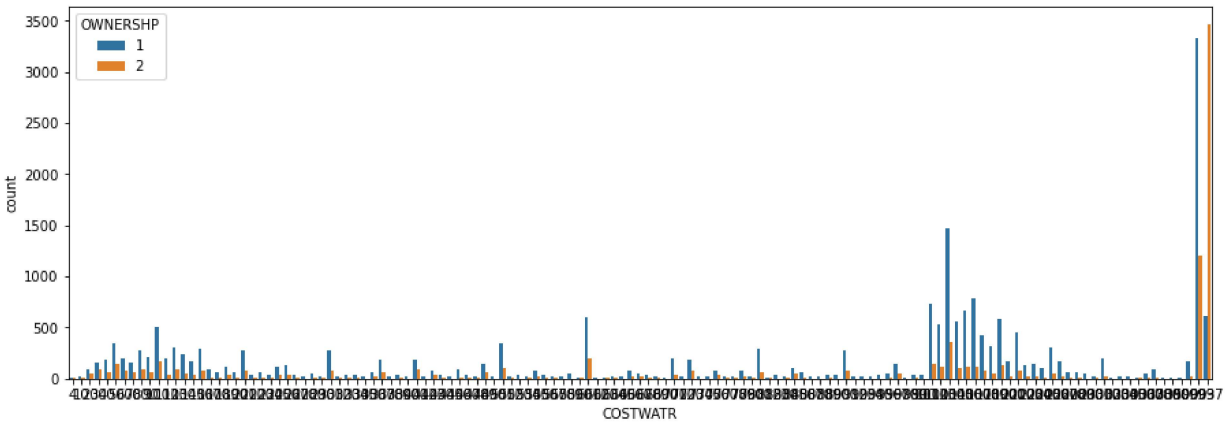
Out[46]:

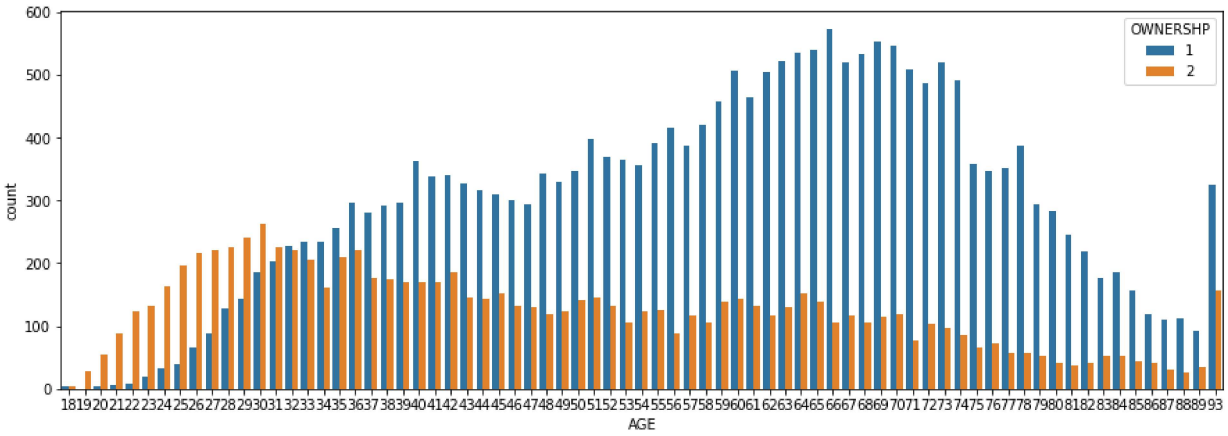
	feature	importance
0	COSTWATR	0.045967
1	AGE	0.039296
2	BEDROOMS	0.037494
3	ROOMS	0.027349
4	HHINCOME	0.013991
5	DENSITY	0.012579
6	COSTGAS	0.008213
7	MARST_Married SP	0.007596
8	NCOUPLES	0.003782
9	MARST_Never married	0.002386

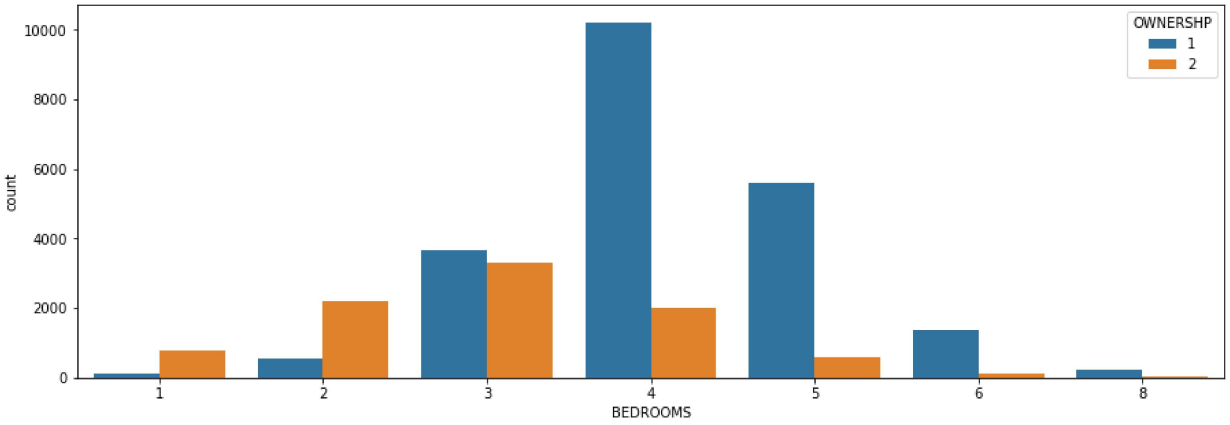
In [47]:
```
importance_df.iloc[:5].plot(kind='bar', x='feature', y='importance')
plt.title('Feature Importances')
plt.show()
```

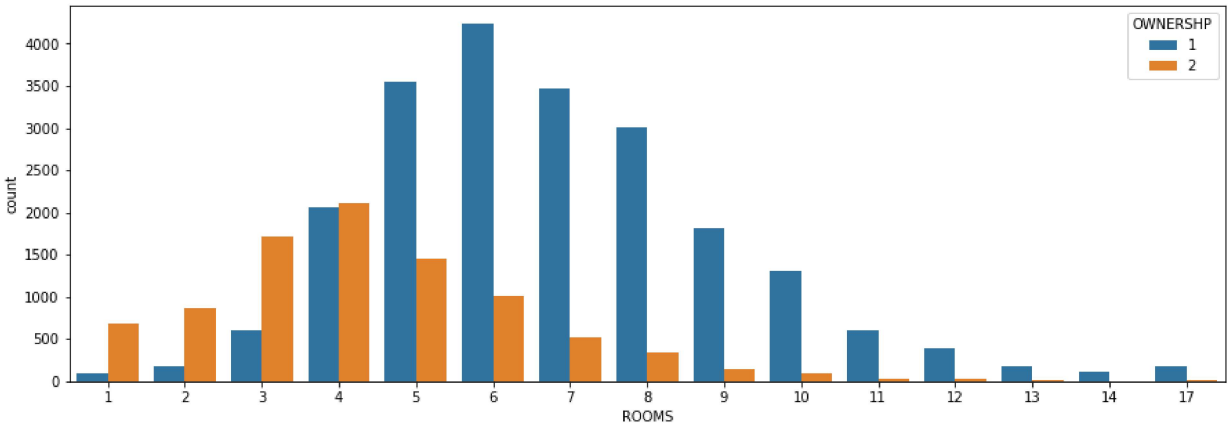


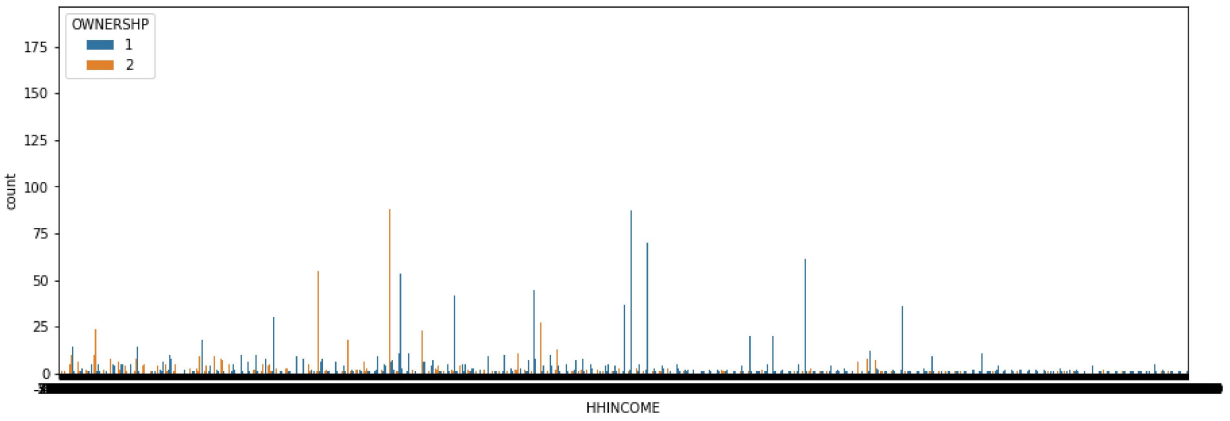
In [48]:
```
import seaborn as sns

# Get the top 5 important variables
top_5_vars = importance_df.iloc[:5]['feature'].tolist()

# Plot countplots for each of the top 5 variables
for var in top_5_vars:
    plt.figure(figsize=(15, 5))
    sns.countplot(x=var, hue='OWNERSHP', data=data)
    plt.show()
```











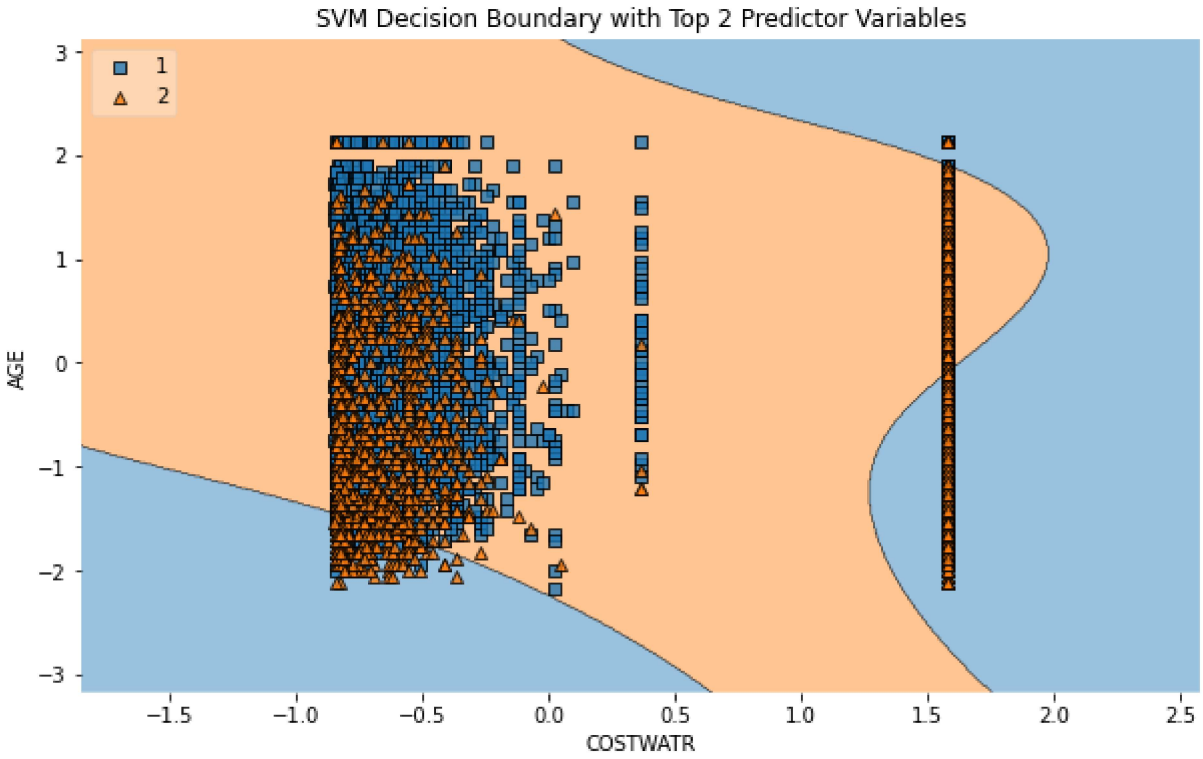
In [49]:
```
# Get the top 2 predictor variables
top_predictors = importance_df.head(2)['feature'].values
print("Top 2 predictor variables:", top_predictors)
```

```
Top 2 predictor variables: ['COSTWATR' 'AGE']
```

In [50]:
```
# Filter the dataset to keep only the top 2 predictor variables
X_top_predictors = X[top_predictors]

# Split the data into training and testing sets
X_train_top, X_test_top, y_train_top, y_test_top = train_test_split(X_top_predictors,

# Scale the training and testing data
scaler_top = StandardScaler()
X_train_top_scaled = scaler_top.fit_transform(X_train_top)
X_test_top_scaled = scaler_top.transform(X_test_top)

# Train the best model with the top 2 predictor variables
```

```
poly_svc_top = SVC(kernel='poly', degree=4 , coef0=1, C=0.1 , gamma='scale', cache_siz
poly_svc_top.fit(X_train_top_scaled, y_train_top)

# Plot the decision boundary
plt.figure(figsize=(10, 6))
plot_decision_regions(X_test_top_scaled, y_test_top.to_numpy(), clf=poly_svc_top, lege
plt.xlabel(top_predictors[0])
plt.ylabel(top_predictors[1])
plt.title('SVM Decision Boundary with Top 2 Predictor Variables')
plt.show()
```

```
[LibSVM]
```



In [ ]:

In [ ]: