Exploring Homeownership Prediction using Support Vector Models: A Case Study of Washington State

Zahra Ahmadi Angali

Abstract

This report investigates the use of support vector models to predict homeownership with the use of demographic and housing data from Washington State. The dataset includes variables about individuals' characteristics and housing features, such as age, income, education level, and the value of the house, with an application of a linear, radial, polynomial kernel in Support Vector Models. Our objective is to predict whether a dwelling is occupied by owners or renters, through using exploratory data analysis, preprocessing, and modeling, key predictors of homeownership are identified. The linear model obtained 84.62% accuracy on the test set using its own respective subset of top ten predictors, the model with the radial kernel gave an accuracy of 76.22% on its respective subset of top ten predictors, and the polynomial model performed with an accuracy of 85.12% on its respective subset of top ten predictors. It can be stated that in all three models the variable that represents the age of the dwelling occupying the house as the characteristic feature and the number of bedrooms as the housing feature , is presented to be of importance in predicting whether the house is rented or owned. Through the analysis conducted in this research, valuable insights can be provided to policymakers and stakeholders in the housing sector.

Introduction

In this project, we investigate the use of SVM classifiers on a housing dataset sourced from Washington State, obtained through the US Census, and accessed via IPUMS USA. This dataset consists of an extensive range of variables, including demographic information about individuals as well as housing-related attributes. The Variables include personal demographics such as age, income, and education level, as well as housing-related metrics such as electricity costs, built year of the house, and surrounding population density. Our primary goal is to predict whether a dwelling is occupied by owners or renters using these factors, therefore the response variable is set to be the one indicating the house is owned or rented and a handful of variables is chosen to be the predictor variables, through the use of different SVM models with kernels like the linear, Polynomial and Radial.

Theoretical Background

Support Vector Machine(SVM):

Support Vector Machines (SVMs) are an effective type of supervised learning algorithms used mainly for classification tasks. SVMs work on the principle of creating an optimal hyperplane in a high-dimensional space to distinguish between different classes. This hyperplane is positioned to increase the margin, which is the distance between the hyperplane and the nearest data points in each class. By increasing this margin, SVMs are naturally prone to overfitting and can generalize well to new data.

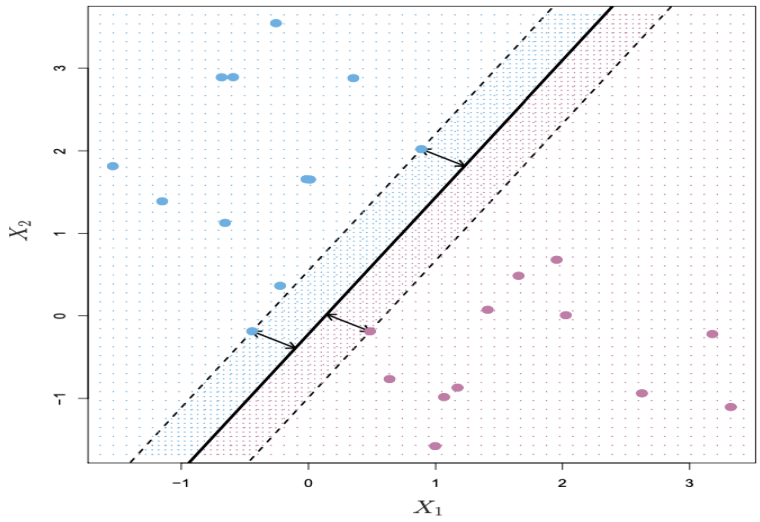

Figure 1 : Decision boundary For a SVM model with Linear kernel

Linear:

For Support Vector Machines with a linear kernel, the decision boundary between classes is a straight line or a hyperplane. The linear kernel is the most basic kernel function used in SVMs. It calculates the dot product of the input data vectors. When the data is linearly separable, which means that the classes can be separated by a straight line or hyperplane, the linear kernel is a better choice. The SVM algorithm determines the optimal hyperplane that maximizes the margin between the classes, which is the distance between the hyperplane and the nearest data points from each class. The linear kernel is efficient and works well with linearly separable data. When working with the linear kernel, it is important to tune the cost parameter. The cost parameter governs the trade-off between maximizing margins and minimizing training data point misclassification. A higher cost parameter increases the penalty for misclassified points,

resulting in a more complex decision boundary that better fits the training data but may be overfitting.

Radial:

When dealing with nonlinear data that cannot be separated by a straight line or hyperplane, SVMs can employ the radial basis function (RBF) kernel. The RBF kernel changes the input data into a higher-dimensional space, potentially infinite-dimensional, in which the classes are linearly separable. This kernel function determines the similarity of two data points based on their distance from one another. Data points that are close together in the original space will be mapped to similar values in the higher-dimensional space, whereas points that are far apart will be mapped to opposite values. The RBF kernel enables SVMs to identify complex, non-linear decision boundaries that accurately separate classes. The trade-off is that the RBF kernel is less interpretable than the linear kernel, and training the model may require more computational resources. Furthermore, the RBF kernel has parameters that must be tuned, such as the gamma value, which determines the influence of individual training examples. SVMs use the RBF kernel to effectively handle non-linear relationships and achieve high accuracy in classification tasks.

Polynomial:

The polynomial kernel is another option for addressing non-linear relationships in data when using Support Vector Machines. In contrast to the linear kernel, which can only separate classes using a straight line or hyperplane, the polynomial kernel maps the input data into a higher-dimensional feature space, allowing SVMs to find more complex, curved decision boundaries. The polynomial kernel computes the dot product of the input data vectors raised to a specified power, resulting in new features that combine the original features. The degree of the polynomial kernel, a tunable parameter, determines the decision boundary's complexity.

Methodology

The dataset initially containing 24 columns with 75388 observations is examined and preprocessed to prepare it for the SVM models. As we dwell into the data, the variables that are thought to be of the least importance in predicting homeownership are dropped from the dataset. Following are the variables that are thought to be of no importance or are causing data redundancy, detailed version of homeownership, cost of fuel number of families, number of couples, number of people, and detailed version of education.

The variable indicating the birth year is dropped because the variable indicating the age of each individual is already present in the house. We can firmly state that the value of the house is of

great importance in indicating whether the house is rented or owned. As for this report, the aim is to explore the influence of other possible variables, therefore the predictor VALEH is also dropped from the dataset. The remaining dataset is examined for the presence of missing or null values, though it is not explicitly demonstrated in the dataset, for each variable a specific value is set as its missing value. Hence as the next step of the preprocessing stage, the specific missing value indicator is identified for its respective variable and filtered from the dataset.

As we proceed with the preprocessing it is noticed that marital status is divided into six categories, to distinguish the influence of each category of Married, divorced, and single on their home ownership status, each subcategory is given as a binary variable in a distinct column. Furthermore, the primary renters or the owners in a household need to be identified, for this step the variables 'SERIAL' demonstrating the serial number of each house are sorted using the 'AGE' variable, and duplicates for the serial variable are also removed from the data set. As a result of this step, we have unique serial values set for the oldest-aged person present at each house. Since the serial number of each house is no longer of use in predicting whether a house is owned or rented therefore its dropped.

With implementing the above steps the dataset is decreased to 15 columns with 28955 observations. When performing a machine learning model it is essential to evaluate the performance of the model, hence the remaining data set is split into training and testing subsets, 70% into a training set providing the model a sufficient amount of data for it to learn and the rest assigned to a testing subset, with this we can assess a model's performance and generalizability. In the last step of the preprocessing stage, the variables are scaled to improve the stability, performance, and interpretability of machine learning models.

Linear:

The analysis is proceeded by fitting a Principal Component Analysis on the scaled training set and through GridSearchCV evaluated the accuracy for various cost values. PCA can be used as a preprocessing step to decorrelate features and remove redundant data prior to training an SVM model. Decorrelating features can improve the performance of linear SVMs by reducing variability and ensuring that the decision boundary is unaffected by the data's correlation structure. With the accuracy of the PCA-modified model being low it can be concluded that PCA is not a right fit to implement on this set of data. In search of obtaining the optimal cost value, once again GridSearchCV is implemented on an SVM model with a **Linear** kernel that has a random cost value, the aim is to acquire the best-performing cost that would give the highest accuracy. With the result of the best-performing cost value, I performed **Recursive Feature Elimination** (RFE) that eliminates the least important variables. Linear SVM models assign coefficients to each feature, indicating their importance in determining the decision boundary. These coefficients represent the contribution of each feature to the decision function.

Using **Permutation importance** on the selected features obtained in the RFE step, Permutation importance is model-agnostic and evaluates feature importance based on the model's prediction performance, top ten variables are presented in descending order. A new subset containing the top ten variables obtained from the permutation importance is created, followed by a new model on the training and a new procedure to obtain the best-performing cost value on this set of data. The accuracy of this final model is computed on the training and the testing set to check whether the model is overfitting or is prone to bias.

The SVM model's decision boundary is illustrated using the two most influential predictors. This boundary provides insight into how the model distinguishes between owned and rented homes, based on the top two prominent characteristics. Indicating the linear relationship of the top two predictors classifies data points into their corresponding classes of owned and rented homes. This implies that the relationship between these predictors and the ownership status is predominantly linear in nature.

Radial:

The analysis for the **Radial** kernel is constructed, by initiating a GridSearchCV to tune the necessary parameters of cost and gamma for the radial kernel on an SVM model with the kernel defined as **Radial** and that has a default cost value of 1 and a default value of 1 for gamma. The aim is to acquire the best-performing cost and gamma value that would give the highest accuracy. With the result of the best-performing cost value, a **Recursive Feature Elimination** is performed (RFE) that eliminates the least important variables.

Using **Permutation importance** on the selected features obtained in the RFE step, Permutation importance is model-agnostic and evaluates feature importance based on the model's prediction performance, top ten variables are presented in descending order. A new subset containing the top ten variables obtained from the permutation importance is created, followed by a new model on the training and a new procedure to obtain the best-performing cost value on this set of data. The accuracy of this final model is computed on the training and the testing set to check whether the model is overfitting or is prone to bias.

At the end of the Radial kernel analysis, the SVM model's decision boundary is illustrated using the two most influential predictors. This boundary provides insight into how the model distinguishes between owned and rented homes, based on the two prominent characteristics. It can effectively classify the data points into their corresponding classes of owned and rented when the predictors and the target variable have a non-linear relationship. Since the model creates clusters of data points, implying the model’s adaptability to varying densities and distributions of data points, leads to potentially improved classification accuracy.

Polynomial:

Exploration of the data using the polynomial kernel is preceded, by initiating a GridSearchCV for parameter tuning on an SVM model with a **Polynomial** kernel that has a default cost value of 1 and default degree of 2. With tuning, the aim is to acquire the best-performing cost along with the optimal degree that would give the highest accuracy. With the result of the best-performing cost value, I performed **Recursive Feature Elimination** (RFE) that eliminates the least important variables.

Using **Permutation importance** on the selected features obtained in the RFE step, Permutation importance is model-agnostic and evaluates feature importance based on the model's prediction performance, top ten variables are presented in descending order. A new subset containing the top ten variables obtained from the permutation importance is created, followed by a new model on the training and a new procedure to obtain the best-performing cost value on this set of data. The accuracy of this final model is computed and demonstrated through the ROC figure on the training and the testing set to check whether the model is overfitting or is prone to bias.

Followed by an illustration of the SVM model's decision boundary using the two most influential predictors. This boundary provides insight into how the model distinguishes between owned and rented homes, based on these two prominent characteristics based on the non-linear relationship of the two predictors and the target variable implying a more complex relationship.

Computational Results

Linear

To find the best performing SVM model with linear kernel the hyperparameter needs to be tuned, therefore through the implementation of grid search I evaluated the accuracy of different cost values, as shown in the figure below we can understand among the ranges of 0.01 to 100 for given for the cost value 10 has the best performance through the representation of the highest accuracy of Mean Test score among the all the values, which in this scenario the highest score is at 47.85%.

Test Accuracy initial SVM model with C=10:	**83.40%**
Train Accuracy initial SVM model with C=10:	**82.85%**

Moreover, the analysis with the obtained cost value to identify the most influential factors on the target variable. The top ten elements are identified through the implementation of **RFE** and Permutation importance with the cost value is once again evaluated, resulting 1 having the best performance. The accuracy of the test and the training set of the new model with the best cost value for the new subset of data are as follows.

Accuracy on the training set	**82.86%**
Accuracy on the testing set	**82.40%**

Presenting that the model is performing well since both the accuracy on the training and test are high and are almost the same.

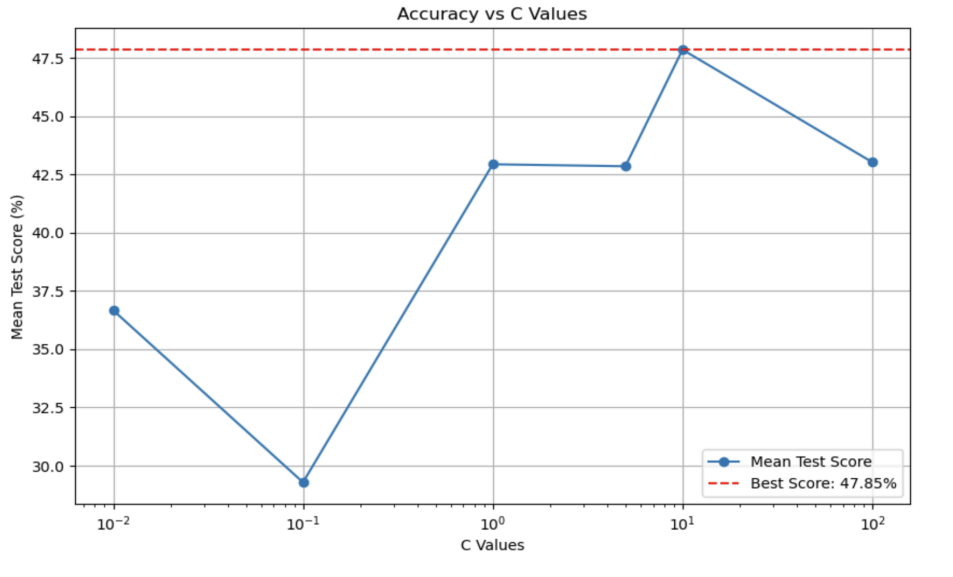

Figure 2: Accuracy vs C values for the initial linear model

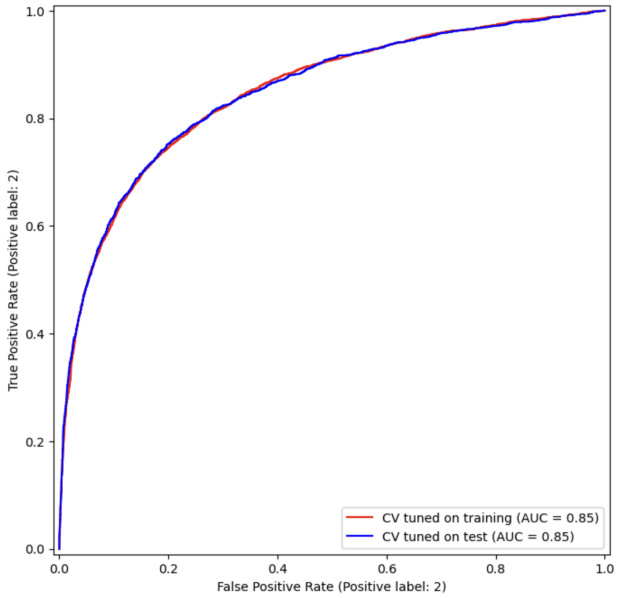

Figure 3: ROC curve

The ROC curve demonstrated how well the model can differentiate between the two classes by comparing the rate of correctly predicted positive instances (True Positives) to the rate of incorrectly predicted negative instances (False Positives). As our model, a perfect model would have a ROC curve that hugs the top-left corner, indicating high sensitivity and a low false positive rate.

I continued the linear kernel SVM model exploration by conducting a model on the top two variables age of the people occupying the homes and the number of bedrooms of the house. The results are presented in the table below.

Test Accuracy top two predictors linear SVM model with C=1	**81.20%**
Train Accuracy top two predictors linear SVM model with C=1	**81.81%**

Radial

With the implementation of GridSearchCV on a Radial kernel model with random values for cost and gamma resulting in an accuracy score of 83.03% with a cost of 1 and gamma of 0.5.

Test Accuracy initial SVM model with gamma=0.5, C=1	**83.27%**
Train Accuracy initial SVM model with gamma=0.5, C=1	**86.99%**

The new model went through RFE and Permutation importance in search of the top ten influential variables.

The GridSearch on the new sub-set of the data presents a cost value of 100 and gamma of 0.5 giving the best accuracy. Through tuning the hyperparameters cost and gamma the accuracy of the training and testing is decreased to the following.

Radial SVM with C=100 gamma=0.5 on test set	**76.22%**
Radial SVM with C=100 gamma=0.5 on train set	**87.06%**

We can conclude that, despite having a higher cost value, the accuracy of the subset model has decreased on both the test and training sets than the initial model. This suggests that the subset model is overfitting the training data, as demonstrated by the significant drop in accuracy on the test set. We can argue, the decrease in accuracy on the train set suggests that the subset model may have lost some important information during feature selection, resulting in a decrease in the model's generalizability.

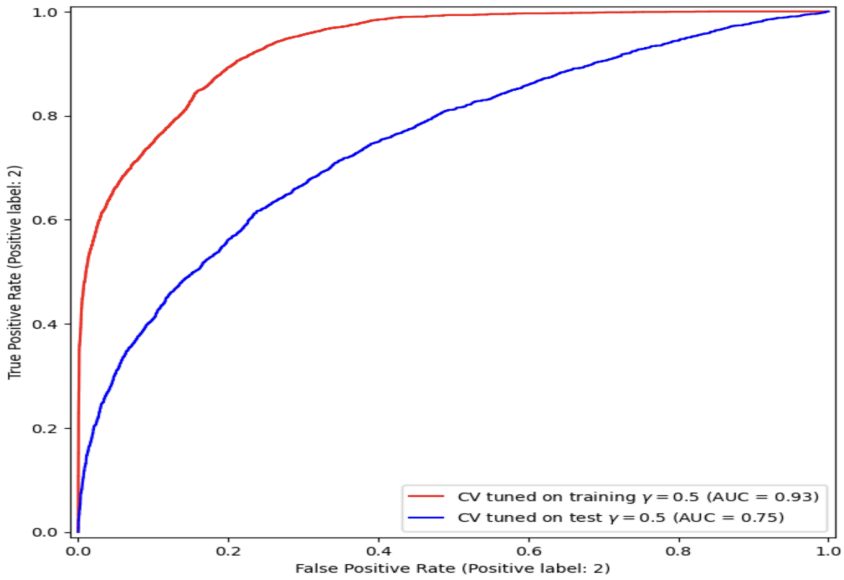

Figure 4: ROC curve

From the ROC curve figure showcased we can see the difference in the accuracy of train and test. With the curve of the train being in the far left. However, the testing model is a little more towards the center, indicating the model's performance is moderate. Overall the accuracy of the train being much higher than the test demonstrates the model's inability to generalize to the test set.

I continued the Radial kernel SVM model exploration by conducting a model on the top two variables of number of bedrooms of the home and age of people occupying the homes . The results presented in the table below show an increase in the accuracy of the test set compared to the 10-variable subset model.

Radial SVM top two with gamma=0.5, C=100 on test set	**82.16%**
Radial SVM top two with gamma=0.5, C=100 on train set	**81.91%**

Polynomial

Lastly the analysis with the polynomial kernel, after conducting a GridSearchCV on a model with default values of cost of 1 and a degree 2. The best combination for the polynomial is set to be the cost of one with a degree of 4. Following are the results of the accuracy of the model on the train and the testing set.

Test Accuracy initial SVM model with degree=4, C=1	**85.36%**
Train Accuracy initial SVM model with degree=4, C=1	**85.39%**

After the implementation of the feature selection methods, the top ten variables are drawn out from the original subset followed by identifying the optimal hyperparameters for the model with the latest subset.

Polynomial SVM with C=1 and degree=4 on the test set	**85.12%**
Polynomial SVM with C=1 and degree= 4 on the train set	**84.95%**

There is a slight decrease but not significant enough for us to question the performance of the model, with the accuracy of the training and testing being close to one another and reflecting the model is performing well.

I continued the Polynomial kernel SVM model exploration by conducting a model on the top two variables of number of bedrooms and age of people occupying the homes. The results are presented in the table below.

Polynomial SVM top two with degree=4, C=1 on the test set	**78.39%**
Polynomial SVM top two with degree=4, C=1 on train set	**77.46%**

With the decrease in the accuracy of the model we can indicate that the model is not performing well due to loss of information.

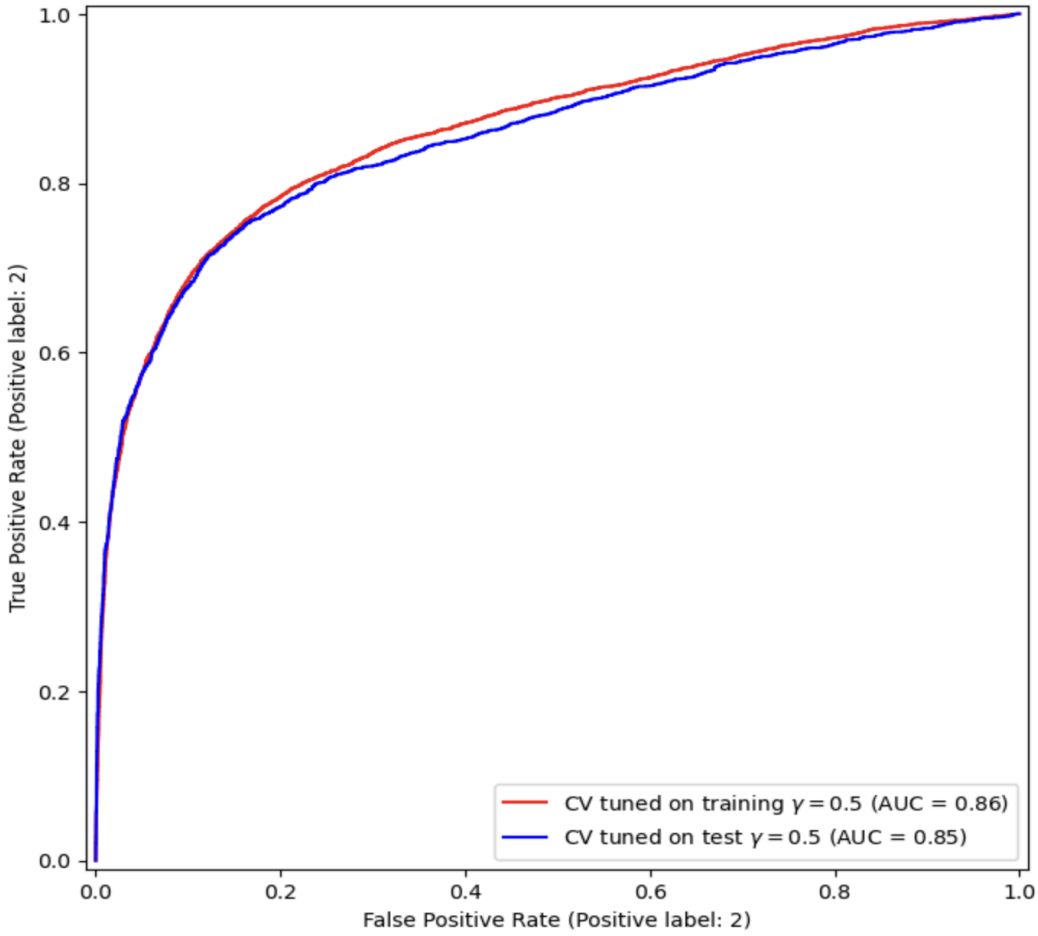

Figure 5: ROC curve

The ROC curve above demonstrates the model's ability to distinguish between the two classes by comparing the rate of correctly predicted positive instances (True Positives) to the rate of incorrectly predicted negative instances (False Positives).

Discussion

In this research, the main objective was to predict the factors influencing the homeownership status using demographic and housing data from Washington State. The investigation took place through conducting Support Vector Machine (SVM) models with various kernel Linear, Radial, and Polynomial. Before proceeding with the analysis stage it was brought to attention the influence of the variable indicating the value of home on the dwelling's tendency to rent or own the house. As a result, the variable was removed from the subset in order to investigate other demographic and characteristic variables.

The Linear SVM model demonstrated decent performance, achieving an accuracy of approximately 84% on the test set. Feature selection techniques, including Recursive Feature Elimination (RFE) and Permutation Importance, identified the top predictors influencing homeownership status. Interestingly, variables such as age and bedroom count emerged as significant predictors across different SVM kernels, indicating their consistent importance in predicting homeownership. Below are presented the top ten variables in descending order from the most influential to the least.

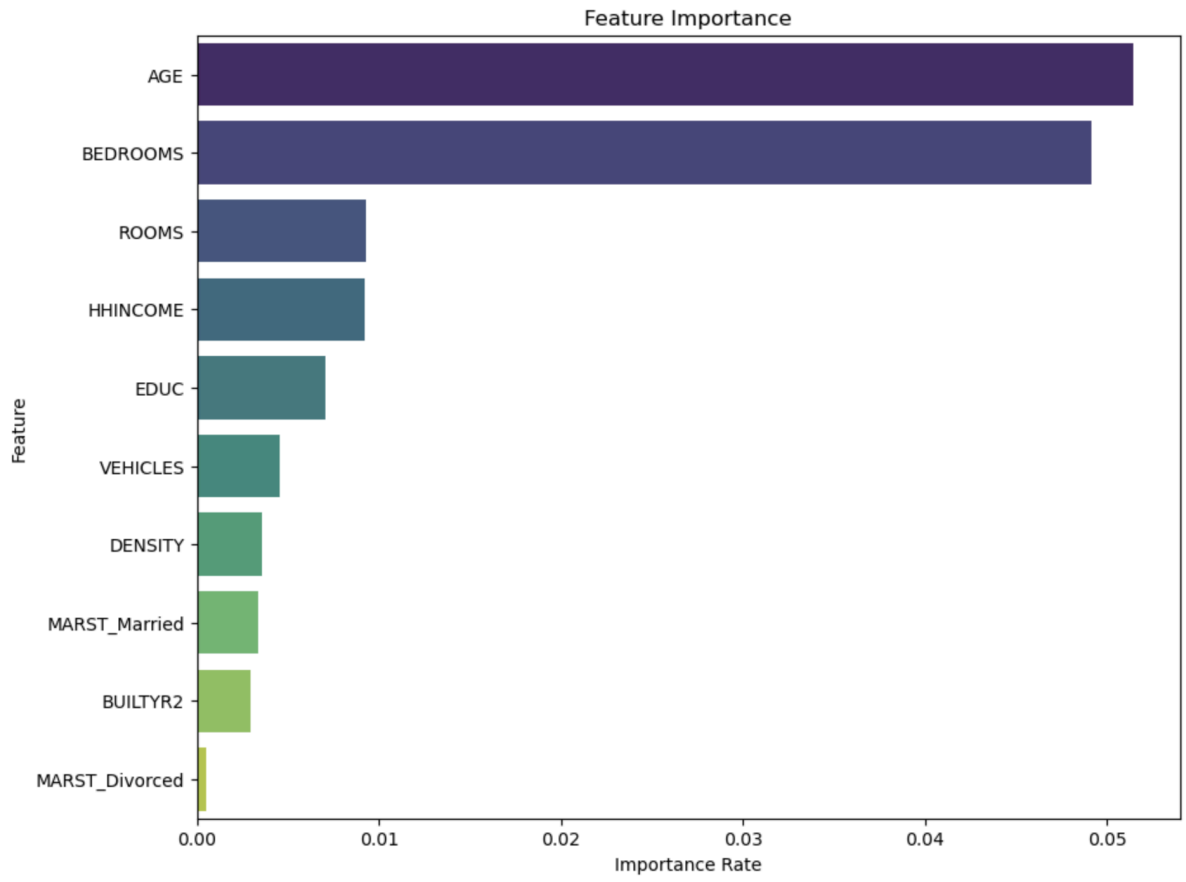

Figure 6 : selected top features for the linear model

With the continuation of the linear kernel SVM model on the top two predictors age and bedroom, there was a decrease in the accuracy of the model though it is still performing well with an accuracy of 81% for both training and testing we can state that the top two predictors do in fact play a crucial role in predicting the ownership status but with more variables, we could make a more accurate prediction.

From the above figure, it can be argued that older individuals are more likely to own a home with more bedrooms and more rooms as their income allows them to make decisions like owning a home. As variables such as age number of bedrooms and rooms and income are shown as the top predictors for the model with a linear kernel

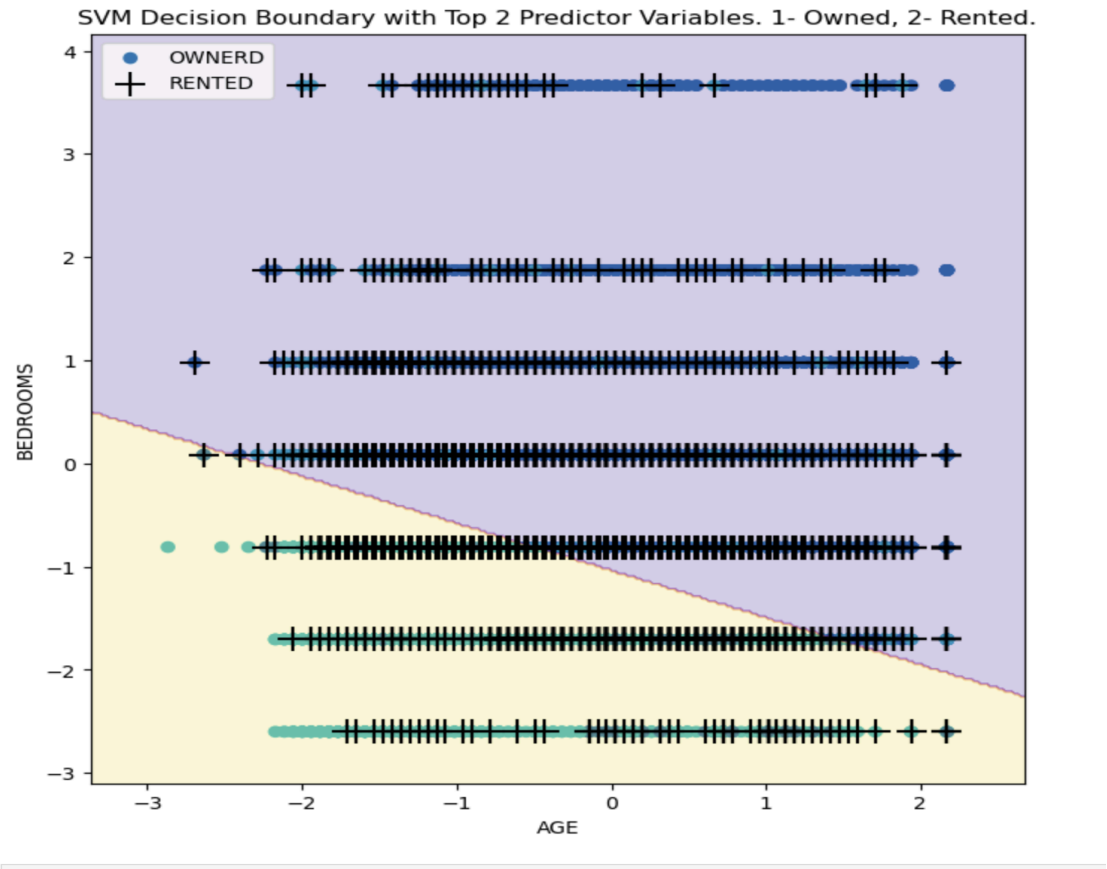

Figure 7 : Linear decision boundary for the top two predictors

For a model that predicts homeownership based on the number of bedrooms and age, the decision boundary figure shows how the SVM separates owned and rented homes using a straight line in the feature space defined by these two predictors. This boundary divides the dataset optimally, maximizing the difference between owned and rented homes and allowing for accurate classification of new homes based on number of bedrooms and age. The ones that have a higher age with more bedrooms are shown to be owners and we can see some data points indicating the dwellings that are younger with homes that have less number of bedrooms are to be owners as well the data points in the middle are more likely to be renters.

The Radial SVM model exhibited slightly better performance with the presence of overfitting indicating high variance and low bias, with an accuracy of around 85% on the test set being less than the accuracy of 91% on the training set. Though feature selection and hyperparameter tuning did result in slightly lower variance, it led to a decrease in accuracy, maintaining the concerns about potential information loss or overfitting. Hence, the model's ability to generalize to unseen data decreased, as evidenced by the lower accuracy on the test set.

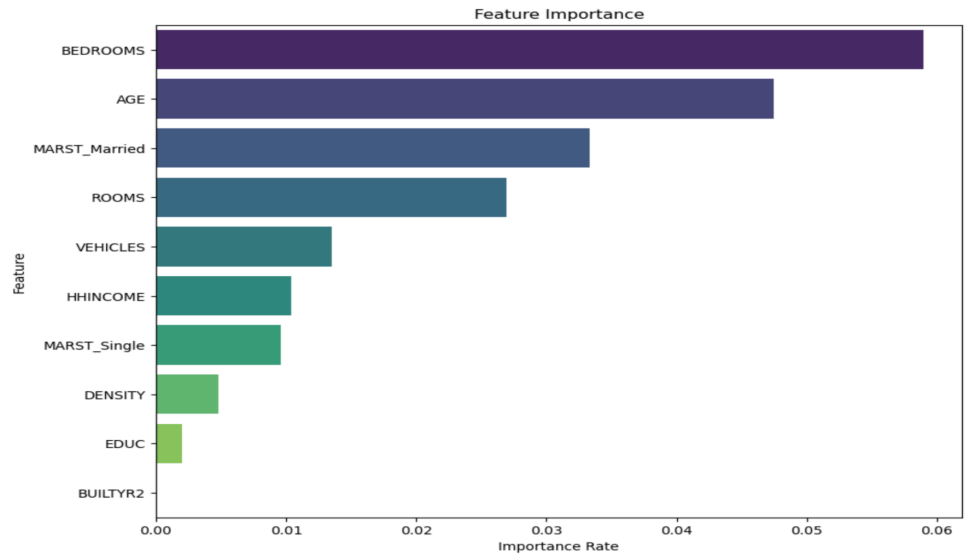

Figure 8 : selected the top features for the Radial model

While the model using the top two predictors, number of bedrooms and age, may not be as accurate as the initial model, with an accuracy of almost 82% for both train and test it exhibits better generalization and provides a more straightforward interpretation of the factors influencing homeownership prediction. It can be drawn from the above figure that the number of bedrooms precedes to be the top indicator followed by the variable indicating the age of individuals, contrary to the linear kernel. Demonstrating the number of bedrooms and age of the individuals along with them being married plays a crucial role in wanting to own a house.

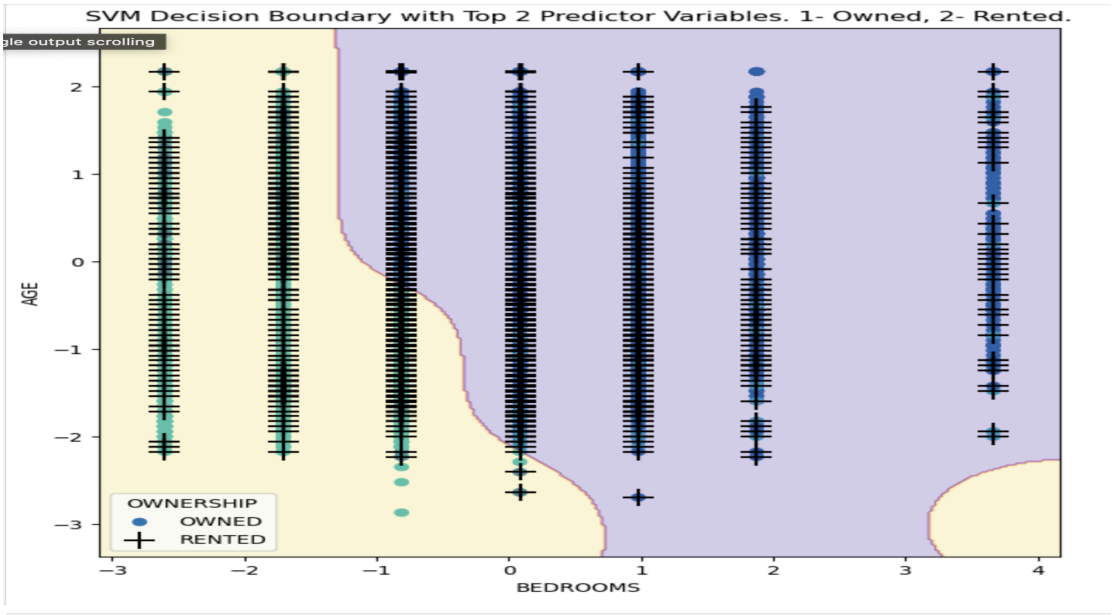

Figure 9:: Radial decision boundary for the top two predictors

The radial kernel decision boundary figure illustrates how a model that predicts homeownership based on predictors such as the number of bedrooms and age distinguishes between owned and rented homes. The nonlinear boundary demonstrates complex connections between predictors, reflecting how variations in age and number of bedrooms influence the classification outcome, improving the model's accuracy in predicting homeownership status. In this analysis, a great number of renters are being misclassified as owners, and some owners are being misclassified as renters.

Similarly, the Polynomial SVM model achieved comparable accuracy to the Radial SVM, with an accuracy of approximately 85% on the test set. Despite slight fluctuations in accuracy after feature selection and hyperparameter tuning, the model maintained relatively consistent performance on both training and testing datasets.

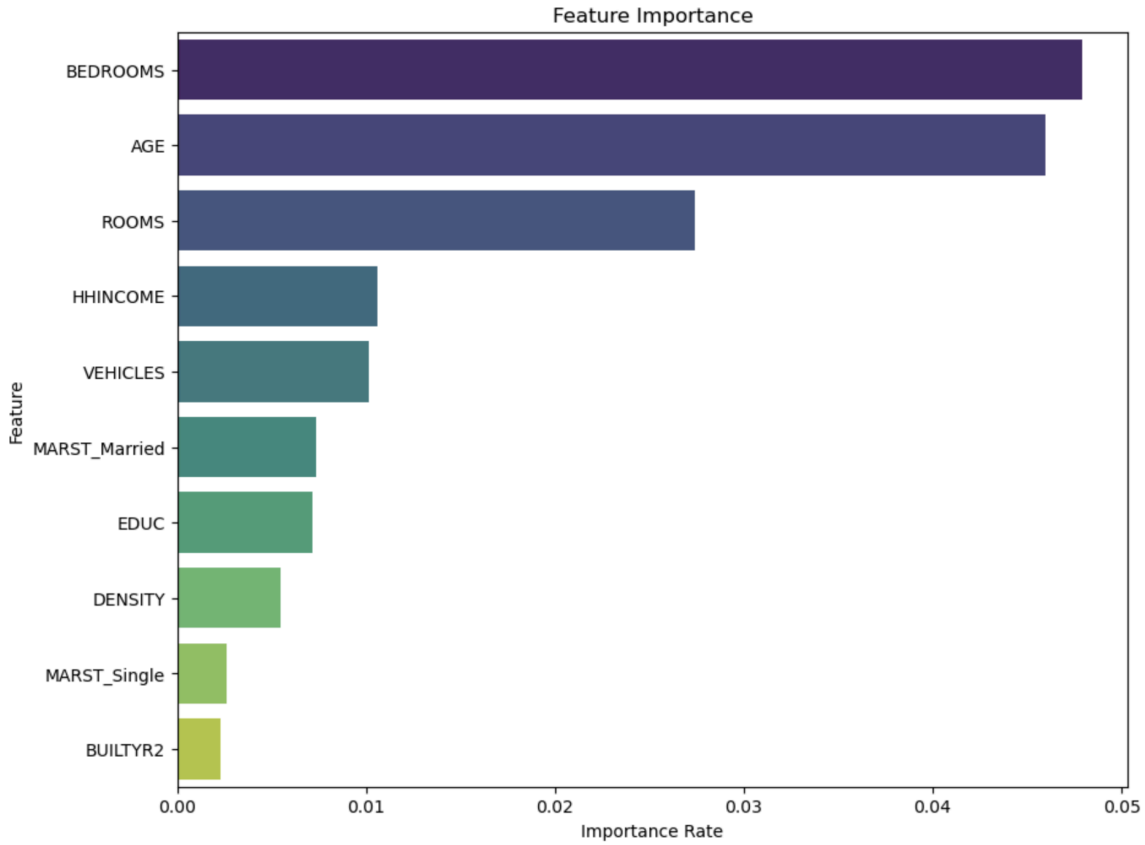

Figure 10: selected the top features for the Polynomial model

With the analysis of the top two variables bedrooms and age the stable accuracies of both the train and test decreased to 78% for the test and 76% for the train demonstrating the crucial influence of the other predictors. Stating that only the top two variables are not sufficient for predicting the homeownership status.

It can be indicated from the figure above that the age of the people living in the homes and the number of bedrooms play a crucial role in people’s decisions to own, along with factors such as

their income the number of vehicles, and their martial statutes to be influencing their choices in owning a house. It can be argued, that individuals with higher income who are older tend to own a home with more bedrooms.

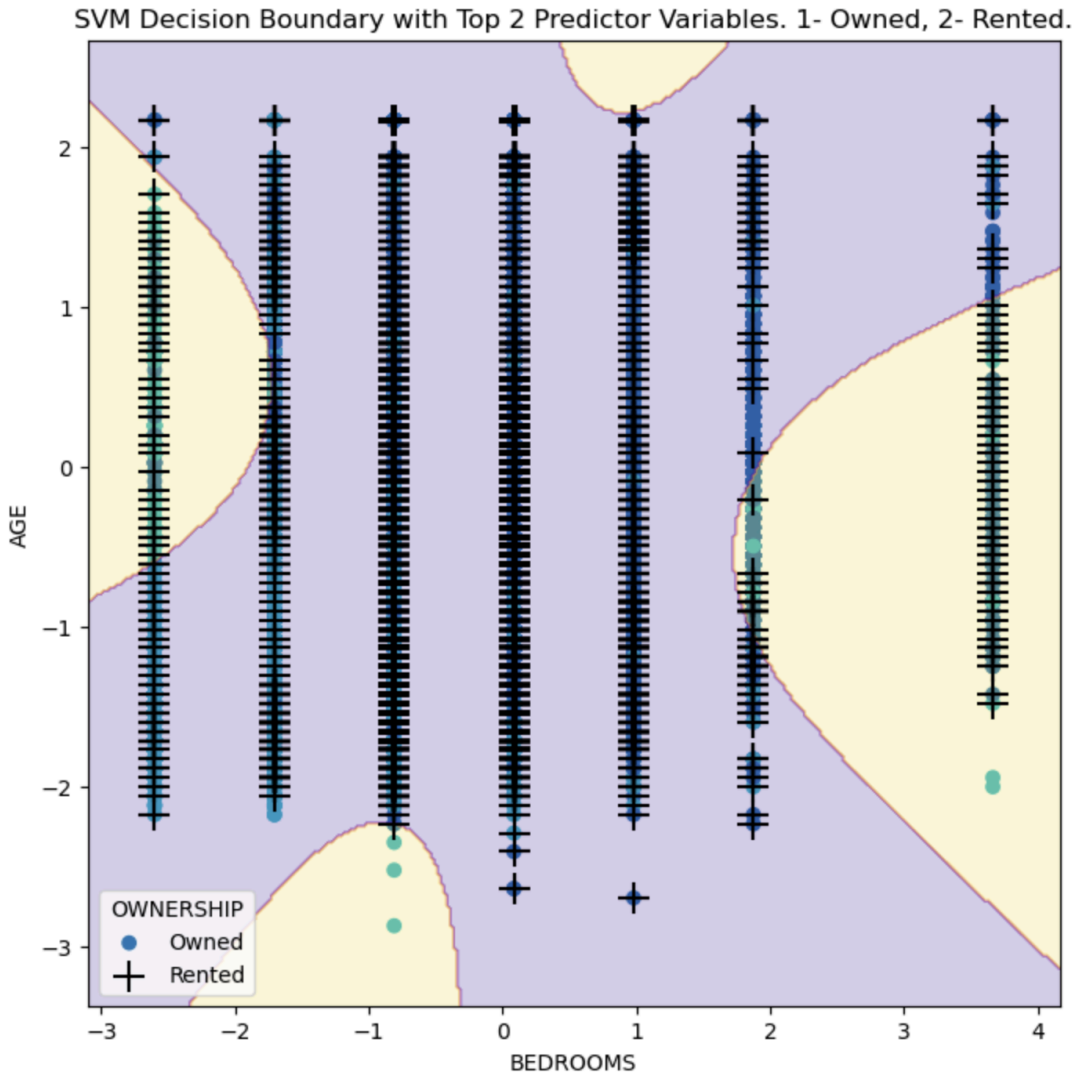

Figure 11: Polynomial decision boundary for the top two predictors

The polynomial kernel decision boundary figure demonstrates the model's approach to predicting homeownership status using the two top features, number of bedrooms and age. The figure above indicates a majority of the renters are being misclassified as owners with fewer owners being misclassified as renters, as this figure gives more boundary to the owners than the renters.

Conclusion

In conclusion, SVM models show promise in predicting homeownership status using demographic and housing data. While each kernel linear, radial, and polynomial offers its own unique advantages, such as interpretability and flexibility, the choice of kernel should be determined by the specific characteristics of the dataset and the desired trade-offs between bias and variance. With accuracies on the testing set for the models with the following kernel the Linear 83% and the Radial 76% and Polynomial with 85% all indicate promising results on their respective top ten variable subset.

With exploring the top ten variables of the SVM model for all three kernel of Linear, Radial and Polynomial it can be stated that people who are older in age have tendency to own a home with more rooms and bedrooms. And factors such as their marital status of being married and their income also influences their choice of ownership.

The key findings indicate the importance of hyperparameter tuning to optimize model performance and generalizability along with feature selection methods to identify the most influencing features unveiling the importance of the age of individuals and the number of bedrooms in predicting the status of home dwellers. Providing valuable insights for future analysis and policymakers.

References

1. https://usa.ipums.org/usa-action/revisions
2. https://scikit-learn.org/stable/modules/permutation_importance.html **“ “Permutation Importance”**
3. https://ieeexplore.ieee.org/document/5337549 **“ RFE”**
4. **Python Machine Learning Sebastian Raschka and Vahid Mirjalili**
5. https://www.statlearning.com/resources-python **“Decision boundary plot for the Theoretical Background”**

Code Appendix

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
from sklearn.decomposition import PCA
from sklearn.model_selection import GridSearchCV
import sklearn.model_selection as skm
from sklearn.datasets import make_classification
from sklearn.feature_selection import RFE
from sklearn.metrics import accuracy_score
import numpy as np
from sklearn.inspection import permutation_importance
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from ISLP.svm import plot as plot_svm
from matplotlib.pyplot import subplots, cm
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import RocCurveDisplay
from sklearn.ensemble import RandomForestClassifier


import seaborn as sns



data = pd.read_csv("/Users/zahra/Library/CloudStorage/OneDrive-SeattleUniversity/spring2024/ML2/SVM/Housing.csv")

data.shape
#we have 75388 data in 24 columns
data.describe()
data.columns

data = data.drop(['OWNERSHPD','COSTFUEL','BIRTHYR','NFAMS','NCOUPLES', 'NFAMS', 'PERNUM','PERWT', 'EDUCD', 'INCTOT', 'VALUEH'], axis=1)

data.columns
data.shape
```

```
conditions = (
    (data['OWNERSHP'] != 0) &
    (data['ROOMS'] != 00) &
    (data['BEDROOMS'] != 00) &
    (~data['EDUC'].isin([00, 99])) &
    (data['BUILTYR2'] != 00) &
    (data['MARST'] != 9)&
    (~data['VEHICLES'].isin([0,9]))&
    (~data['AGE'].isin([999]))

)

'''when trying to get rid of the missing values or the NA for the value of the house variable
causing the variable ownership to loose its 2 value therefore we can sate that when there is
missing in the vaLUE OF THE House,its rented'''

# Apply filtering
data = data[conditions]
data.shape
data.head()
data['MARST'] = data['MARST'].replace({1: 'Married', 2: 'Married', 3: 'Divorced', 4: 'Divorced',
5:'Single', 6: 'Single'})
data = pd.get_dummies(data, columns=['MARST'])
data.columns
data = data.sort_values(['SERIAL', 'AGE'], ascending=[True, False]).drop_duplicates('SERIAL')
data.shape
data.isna().sum()
data=data.drop(['SERIAL'], axis=1)
data.OWNERSHP.unique()
print(data.head())
X = data.drop(['OWNERSHP'], axis =1 )

y = data['OWNERSHP']
#spliting the data into train and test
X_train, X_test, y_train, y_test =  train_test_split(X, y, train_size = 0.7, test_size = 0.3
                                    ,random_state = 1)
scaler = StandardScaler().fit(X_train)
X_trainScaled = scaler.transform(X_train)
X_testScaled = scaler.transform(X_test)
n_components = range(1, 15)

# Initialize variables to store the best accuracy and corresponding n_components
best_accuracy = 0
best_n_components = None
```

```
# Iterate over each value of n_components
for n in n_components:
    # Apply PCA for feature selection
    pca = PCA(n_components=n)
    X_train_pca = pca.fit_transform(X_trainScaled)
    X_test_pca = pca.transform(X_testScaled)

    # Train SVM classifier
     svm_classifier = SVC(kernel='linear',C=10, cache_size=1000, verbose=True, max_iter=1000,
random_state=1)
    svm_classifier.fit(X_train_pca, y_train)

    # Make predictions on test data
    y_pred = svm_classifier.predict(X_test_pca)

    # Calculate accuracy
    accuracy = accuracy_score(y_test, y_pred) * 100

    # Update best accuracy and best_n_components if necessary
    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_n_components = n

# Print the best n_components and corresponding accuracy
print("Best n_components:", best_n_components)
print("Best Accuracy:", best_accuracy)
C_values = [0.01, 0.1, 1, 10, 100, 1000]

svm_mod=  [(C,  SVC(kernel='linear',  C=C,  cache_size=1000,  verbose=True,  max_iter=1000,
random_state=1)
           .fit(X_train_pca, y_train)
           .score(X_test_pca, y_test) * 100)
          for C in C_values]

for C, accuracy in results:
    print("Cost value: %s, Accuracy: %.2f%%" % (C, accuracy))
# Define the list of cost values
C_values = [0.01, 0.1, 1, 10, 100, 1000]

# Define the parameter grid for GridSearchCV
param_grid = {'C': C_values}

# Create an SVM classifier with linear kernel
```

```
svm_classifier = SVC(kernel='linear', cache_size=1000, verbose=True, max_iter=1000,random_state=1)

# Create GridSearchCV object
grid_search = GridSearchCV(estimator=svm_classifier, param_grid=param_grid, cv=5)

# Fit the GridSearchCV object to the data
grid_search.fit(X_train_pca, y_train)

# Get the best parameters and best accuracy
best_params = grid_search.best_params_
best_accuracy = grid_search.best_score_ * 100

# Print the best parameters and best accuracy
print("Best Parameters:", best_params)
print("Best Accuracy: %.2f%%" % best_accuracy)
#with mplimenting pca the accuracy is lowering so we can try some other method the method we can use in the following would be pca
C_values = [0.01, 0.1, 1, 10, 100]

# Define the parameter grid for GridSearchCV
param_grid = {'C': C_values}

# Create an SVM classifier with linear kernel
svm_classifier = SVC(kernel='linear', cache_size=1000, verbose=True, max_iter=1000,random_state=1)

# Create GridSearchCV object
grid_search = GridSearchCV(estimator=svm_classifier, param_grid=param_grid, cv=5)

# Fit the GridSearchCV object to the data
grid_search.fit(X_trainScaled, y_train)

# Get the best parameters and best accuracy
best_params = grid_search.best_params_
best_accuracy = grid_search.best_score_ * 100

# Print the best parameters and best accuracy
print("Best Parameters:", best_params)
print("Best Accuracy: %.2f%%" % best_accuracy)
svm_linear = SVC(C=10, kernel='linear',cache_size=1000, verbose=True, max_iter=1000, random_state=42)
svm_linear.fit(X_trainScaled, y_train)
```

```
kfold = skm.KFold(5,
                  random_state=42,
                  shuffle=True)
grid = skm.GridSearchCV(svm_linear,
                        {'C':[0.01,0.1,1,5,10,100]},
                        refit=True,
                        cv=kfold,
                        scoring='accuracy')



grid.fit(X_trainScaled, y_train)
grid.best_params_
grid.best_estimator_
best_params = grid.best_params_
best_accuracy = grid.best_score_ * 100

# Print the best parameters and best accuracy
print("Best Parameters:", best_params)
print("Best Accuracy: %.2f%%" % best_accuracy)

# Define C values and corresponding mean test scores from grid search
C_values = [0.01, 0.1, 1, 5, 10, 100]
mean_test_scores = grid.cv_results_['mean_test_score'] * 100
best_score = grid.best_score_ * 100

# Plotting
plt.figure(figsize=(10, 6))
plt.plot(C_values, mean_test_scores, marker='o', linestyle='-', label='Mean Test Score')
plt.axhline(y=best_score, color='r', linestyle='--', label='Best Score: {:.2f}%'.format(best_score))
plt.title('Accuracy vs C Values')
plt.xlabel('C Values')
plt.ylabel('Mean Test Score (%)')
plt.xscale('log')  # Logarithmic scale for better visualization
plt.legend()
plt.grid(True)
plt.show()

# Initialize and train the SVM classifier
svm = SVC(kernel='linear', C=0.1, random_state=42)
svm.fit(X_trainScaled, y_train)

# Perform Recursive Feature Elimination (RFE)
```

```
rfe = RFE(estimator=svm, n_features_to_select=10, step=1)
rfe.fit(X_trainScaled, y_train)

# Get selected features
selected_features = np.array(range(X_trainScaled.shape[1]))[rfe.support_]

# Extract feature names
selected_feature_names = X.columns[selected_features]

# Train SVM classifier with selected features
svm_selected = SVC(kernel='linear', C=0.1, random_state=42)
svm_selected.fit(X_trainScaled[:, selected_features], y_train)

# Evaluate the model on the testing data
accuracy = svm_selected.score(X_testScaled[:, selected_features], y_test)
print("Accuracy with selected features: {:.2f}%".format(accuracy * 100))

# Print important features
print("Important Features:")
for feature in selected_feature_names:
    print(feature)
#the important features are printed and with the rfe the accuracy of the model with those selected features is presented to be 84.57%

accuracy = svm.score(X_testScaled, y_test)
print(" Test Accuracy inital svm model with c=0.1: {:.2f}%".format(accuracy * 100))
accuracy = svm.score(X_trainScaled, y_train)
print(" Train Accuracy inital svm model with c=0.1: {:.2f}%".format(accuracy * 100))

accuracy = svm_selected.score(X_trainScaled[:, selected_features], y_train)
print("Accuracy with selected features: {:.2f}%".format(accuracy * 100))
#the accuracy of the train and the test are close to one another and are high therefore we can conclude the model is performing well
plt.figure(figsize=(10, 6))
sns.barplot(x=svm_selected.coef_[0], y=selected_feature_names, palette='viridis')
plt.title('Feature Importance')
plt.xlabel('Coefficient Value')
plt.ylabel('Feature')
plt.show()

# Compute permutation importance
result = permutation_importance(svm_selected, X_testScaled[:, selected_features], y_test, n_repeats=10, random_state=1)
```

```
# Extract feature names
feature_names = selected_feature_names

# Create dataframe to store feature importance
importance_df = pd.DataFrame({'feature': feature_names, 'importance': result.importances_mean})

# Sort dataframe by importance
importance_df = importance_df.sort_values(by='importance', ascending=False).reset_index(drop=True)

# Plotting
plt.figure(figsize=(10, 8))
sns.barplot(x='importance', y='feature', data=importance_df, palette='viridis')
plt.title('Feature Importance')
plt.xlabel('Importance Rate')
plt.ylabel('Feature')
plt.show()

X_New = data[['AGE', 'BEDROOMS','COSTWATR', 'ROOMS', 'DENSITY', 'MARST_Married', 'HHINCOME', 'COSTGAS', 'VEHICLES', 'EDUC']]

y_New = data['OWNERSHP']
#spliting the data into train and test
X_trainNew, X_testNew, y_trainNew, y_testNew =  train_test_split(X_New, y_New, train_size = 0.7, test_size = 0.3
                                    ,random_state = 1)

scaler = StandardScaler().fit(X_trainNew)
X_trainNewScaled = scaler.transform(X_trainNew)
X_testNewScaled = scaler.transform(X_testNew)
X_New.columns

svm_linear = SVC(C=10, kernel='linear',cache_size=1000, verbose=True, max_iter=1000, random_state=42)
svm_linear.fit(X_trainNewScaled, y_trainNew)


kfold = skm.KFold(5,
                  random_state=42,
                  shuffle=True)
grid = skm.GridSearchCV(svm_linear,
                        {'C':[0.001,0.01,0.1,1,5,10,100]},
                        refit=True,
```

```
                cv=kfold,
                scoring='accuracy')



grid.fit(X_trainNewScaled, y_trainNew)
grid.best_params_
#{'C': 0.01}

linear_svc = SVC(kernel='linear', C=0.01, cache_size=1000, verbose = True, max_iter = 10000,
random_state=42)

# Fit the model to the training data
linear_svc.fit(X_trainNewScaled, y_trainNew)

# Evaluate the model on the testing data and print the accuracy score
accuracy = linear_svc.score(X_testNewScaled, y_testNew)
print("Linear SVM with C=0.01: %.2f%%" % (accuracy * 100))
accuracy = linear_svc.score(X_trainNewScaled, y_trainNew)
print("Linear SVM with C=0.01: %.2f%%" % (accuracy * 100))
#since trian and test are close to one another we can say the model is prforming good
y_hat = linear_svc.predict(X_testNewScaled)
conf = ConfusionMatrixDisplay.from_predictions(y_testNew, y_hat)
conf


X_top = data[['AGE', 'BEDROOMS']]

y = data['OWNERSHP']
#spliting the data into train and test
X_trainTop, X_testTop, y_trainTop, y_testTop =  train_test_split(X_top, y, train_size = 0.7,
test_size = 0.3
                                    ,random_state = 1)

scaler = StandardScaler().fit(X_trainTop)
X_trainScaledTop = scaler.transform(X_trainTop)
X_testScaledTop = scaler.transform(X_testTop)

linearsvc_top = SVC(kernel='linear', C=0.01, cache_size=1000, verbose=True, max_iter=10000,
random_state=1)
linearsvc_top.fit(X_trainScaledTop, y_trainTop)

fig, ax = subplots(figsize=(8,8))
plot_svm(X_trainScaledTop,
```

```
        y_trainTop,
        linearsvc_top,
        ax=ax)
ax.set_xlabel('AGE')
ax.set_ylabel('BEDROOMS')
plt.title('SVM Decision Boundary with Top 2 Predictor Variables. 1- Owned, 2- Rented.')
# Add legend with colors
class_labels = ['OWNED', 'RENTED']

# Plot legend
plt.legend(class_labels, title='OWNERSHIP')

plt.show()


accuracy = linearsvc_top.score(X_testScaledTop, y_testTop)

print(" Test Accuracy top two linear svm model with C=0.01: {:.2f}%".format(accuracy * 100))
accuracy = linearsvc_top.score(X_trainScaledTop, y_trainTop)
print(" Train Accuracy inital svm model with C=0.01: {:.2f}%".format(accuracy * 100))

roc_curve = RocCurveDisplay.from_estimator
fig, ax = subplots(figsize=(8,8))
for (X_, y_, c, name) in zip(
    (X_trainNewScaled, X_testNewScaled),
    (y_train, y_test),
    ('r', 'b'),
    ('CV tuned on training',
     'CV tuned on test')):
   roc_curve(linear_svc,
             X_,
             y_,
             name=name,
             ax=ax,
             color=c)
svm_rbf = SVC(kernel="rbf", cache_size=1000, verbose=True, max_iter=10000,gamma=1, C=1,
random_state=42)
svm_rbf.fit(X_trainScaled, y_train)

kfold = skm.KFold(5,
                  random_state=0,
                  shuffle=True)
grid = skm.GridSearchCV(svm_rbf,
                        {'C':[0.1,1,10,100,1000],
```

```
                     'gamma':[0.5,1,2,3,4]},
                    refit=True,
                    cv=kfold,
                    scoring='accuracy');
grid.fit(X_trainScaled, y_train)
grid.best_params_
grid.best_estimator_
best_params = grid.best_params_
best_accuracy = grid.best_score_ * 100

# Print the best parameters and best accuracy
print("Best Parameters:", best_params)
print("Best Accuracy: %.2f%%" % best_accuracy)
#Best Parameters: {'C': 1, 'gamma': 0.5}
#Best Accuracy: 84.06%

results = grid.cv_results_

# Extracting the mean test scores
mean_test_scores = results['mean_test_score']

# Reshaping the mean test scores to match the grid of C and gamma values
mean_test_scores = np.array(mean_test_scores).reshape(len(c_values), len(gamma_values))

# Plotting the heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(mean_test_scores, annot=True, cmap="YlGnBu", xticklabels=gamma_values,
yticklabels=c_values)
plt.title('Accuracy for different C and gamma values')
plt.xlabel('Gamma')
plt.ylabel('C')
plt.show()

svm_rbf = SVC(kernel='rbf', cache_size=1000, verbose=True, max_iter=10000,gamma=0.5,
C=1, random_state=42)
svm_rbf.fit(X_trainScaled, y_train)


accuracy = svm_rbf.score(X_testScaled, y_test)

print(" Test Accuracy inital svm model with gamma=0.5, C=1: {:.2f}%".format(accuracy * 100))
accuracy = svm_rbf.score(X_trainScaled, y_train)
print(" Train Accuracy inital svm model with gamma=0.5, C=1: {:.2f}%".format(accuracy * 100))
```

```
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)

# Perform Recursive Feature Elimination (RFE)
rfe = RFE(estimator=rf_classifier, n_features_to_select=10, step=1)
rfe.fit(X_trainScaled, y_train)

# Get selected features
selected_features = np.array(range(X_trainScaled.shape[1]))[rfe.support_]

# Extract feature names
selected_feature_names = X.columns[selected_features]

# Train SVM classifier with selected features
svm_selected_rbf = SVC(kernel='rbf', cache_size=1000, verbose=True,
max_iter=10000,gamma=0.5, C=1, random_state=42)
svm_selected_rbf.fit(X_trainScaled[:, selected_features], y_train)

# Evaluate the model on the testing data
accuracy = svm_selected_rbf.score(X_testScaled[:, selected_features], y_test)
print("Accuracy with selected features: {:.2f}%".format(accuracy * 100))

# Print important features
print("Important Features:")
for feature in selected_feature_names:
    print(feature)
accuracy_rbf = svm_selected_rbf.score(X_trainScaled[:, selected_features], y_train)
print("Accuracy with selected features: {:.2f}%".format(accuracy_rbf * 100))
#the accuracy of the train is a higher than the test so which is a good sign

# Compute permutation importance
result_rbf = permutation_importance(svm_selected_rbf, X_testScaled[:, selected_features],
y_test, n_repeats=10, random_state=1)

# Extract feature names
feature_names = selected_feature_names

# Create dataframe to store feature importance
importance_df = pd.DataFrame({'feature': feature_names, 'importance':
result_rbf.importances_mean})

# Sort dataframe by importance
importance_df = importance_df.sort_values(by='importance',
ascending=False).reset_index(drop=True)
```

```
# Plotting
plt.figure(figsize=(10, 8))
sns.barplot(x='importance', y='feature', data=importance_df, palette='viridis')
plt.title('Feature Importance')
plt.xlabel('Importance Rate')
plt.ylabel('Feature')
plt.show()

X_rbf = data[['BEDROOMS','AGE','COSTWATR', 'ROOMS','HHINCOME', 'DENSITY',
'COSTELEC', 'COSTGAS', 'BUILTYR2', 'EDUC']]
X_trainRbf, X_testRbf, y_train, y_test =  train_test_split(X_rbf, y, train_size = 0.7, test_size = 0.3
                                    ,random_state = 1)

scaler = StandardScaler().fit(X_trainRbf)
X_trainRbfScaled = scaler.transform(X_trainRbf)
X_testRbfScaled = scaler.transform(X_testRbf)

svm_rbf=SVC(kernel='rbf', cache_size=1000, verbose=True, max_iter=10000,gamma=0.5, C=1,
random_state=42)
svm_rbf.fit(X_trainRbfScaled, y_train)


kfold = skm.KFold(5,
                  random_state=42,
                  shuffle=True)
grid = skm.GridSearchCV(svm_linear,
                        {'C':[0.1,1,10,100,1000],
                         'gamma':[0.5,1,2,3,4]},
                        refit=True,
                        cv=kfold,
                        scoring='accuracy')



grid.fit(X_trainRbfScaled, y_train)
grid.best_params_
grid.best_params_
grid.best_estimator_
best_params = grid.best_params_
best_accuracy = grid.best_score_ * 100

# Print the best parameters and best accuracy
print("Best Parameters:", best_params)
```

```
print("Best Accuracy: %.2f%%" % best_accuracy)
#it is reported that {'C': 100, 'gamma': 0.5} gives the best parameters for this set off data

radial_svc = SVC(kernel='rbf', cache_size=1000, verbose=True, max_iter=10000,gamma=0.5,
C=100, random_state=42)

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

# Evaluate the model on the testing data and print the accuracy score
accuracy = radial_svc.score(X_testRbfScaled, y_testNew)
print("radial SVM with C=100 gamma=0.5 on test set : %.2f%%" % (accuracy * 100))
accuracy = radial_svc.score(X_trainRbfScaled, y_train)
print("radial SVM with C=100 gamma=0.5 on train set: %.2f%%" % (accuracy * 100))
#the test acc is at 76% and the train acc is at 87%

y_hat = radial_svc.predict(X_testRbfScaled)
conf = ConfusionMatrixDisplay.from_predictions(y_test, y_hat)
conf

X_top2 = data[['BEDROOMS', 'AGE']]


#spliting the data into train and test
X_trainTop2, X_testTop2, y_train, y_test =  train_test_split(X_top2, y, train_size = 0.7, test_size
= 0.3
                                    ,random_state = 1)

scaler = StandardScaler().fit(X_trainTop2)
X_trainScaled_Top = scaler.transform(X_trainTop2)
X_testScaled_Top = scaler.transform(X_testTop2)


radialsvc_top = SVC(kernel='rbf', cache_size=1000, verbose=True,
max_iter=10000,gamma=0.5, C=100, random_state=42)
radialsvc_top.fit(X_trainScaled_Top, y_train)

fig, ax = subplots(figsize=(8,8))
plot_svm(X_trainScaled_Top,
        y_train,
        radialsvc_top,
        ax=ax)
ax.set_xlabel('BEDROOMS')
ax.set_ylabel('AGE')
```

```
plt.title('SVM Decision Boundary with Top 2 Predictor Variables. 1- Owned, 2- Rented.')
class_labels = ['OWNED', 'RENTED']

plt.legend(class_labels, title='OWNERSHIP')

plt.show()

accuracy = radialsvc_top.score(X_testScaled_Top, y_test)
print("radial SVM top two with gamma=0.5, C=100 on test set : %.2f%%" % (accuracy * 100))
accuracy =radialsvc_top.score(X_trainScaled_Top, y_train)
print("radial SVM top two with gamma=0.5, C=100 on train set: %.2f%%" % (accuracy * 100))

#the acc of test is increasing but the acc of train is decreasing when takeing the subset of the 2 best predictors


fig, ax = subplots(figsize=(8,8))
for (X_, y_, c, name) in zip(
    (X_trainRbfScaled, X_testRbfScaled),
    (y_train, y_test),
    ('r', 'b'),
    ('CV tuned on training $\gamma=0.5$',
     'CV tuned on test $\gamma=0.5$')):
   roc_curve(radial_svc,
        X_,
        y_,
        name= name,
        color=c,
        ax=ax)
fig;

svm_poly = SVC(kernel="poly", cache_size=1000, verbose=True, max_iter=10000,degree=2,
C=1, random_state=42)
svm_poly.fit(X_trainScaled, y_train)

kfold = skm.KFold(5,
                  random_state=0,
                  shuffle=True)
grid = skm.GridSearchCV(svm_poly,
                        {'C':[0.1,1,10,100,1000],
                         'degree':[2,3,4,5]},
                        refit=True,
                        cv=kfold,
                        scoring='accuracy');
```

```
grid.fit(X_trainScaled, y_train)
grid.best_params_
grid.best_estimator_
best_params = grid.best_params_
best_accuracy = grid.best_score_ * 100

# Print the best parameters and best accuracy
print("Best Parameters:", best_params)
print("Best Accuracy: %.2f%%" % best_accuracy)
#Best Parameters: {'C': 1, 'degree': 3}
#Best Accuracy: 84.78%

degree_values = [2, 3, 4, 5]
results = grid.cv_results_

# Extracting the mean test scores
mean_test_scores = results['mean_test_score']

# Reshaping the mean test scores to match the grid of C and degree values
mean_test_scores = np.array(mean_test_scores).reshape(len(c_values), len(degree_values))

# Plotting the heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(mean_test_scores, annot=True, cmap="YlGnBu", xticklabels=degree_values,
yticklabels=c_values)
plt.title('Accuracy for different C and degree values')
plt.xlabel('Degree')
plt.ylabel('C')
plt.show()

svm_poly = SVC(kernel='poly', cache_size=1000, verbose=True, max_iter=10000,degree=3,
C=1, random_state=42)
svm_poly.fit(X_trainScaled, y_train)


accuracy = svm_poly.score(X_testScaled, y_test)

print(" Test Accuracy inital svm model with degree=3, C=1: {:.2f}%".format(accuracy * 100))
accuracy = svm_poly.score(X_trainScaled, y_train)
print(" Train Accuracy inital svm model with degree=3, C=1: {:.2f}%".format(accuracy * 100))

rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)

# Perform Recursive Feature Elimination (RFE)
```

```
rfe = RFE(estimator=rf_classifier, n_features_to_select=10, step=1)
rfe.fit(X_trainScaled, y_train)

# Get selected features
selected_features = np.array(range(X_trainScaled.shape[1]))[rfe.support_]

# Extract feature names
selected_feature_names = X.columns[selected_features]

# Train SVM classifier with selected features
svm_selected_poly = SVC(kernel='poly', cache_size=1000, verbose=True,
max_iter=10000,degree=3, C=1, random_state=42)
svm_selected_poly.fit(X_trainScaled[:, selected_features], y_train)

# Evaluate the model on the testing data
accuracy_poly_test= svm_selected_poly.score(X_testScaled[:, selected_features], y_test)
print("Accuracy with selected features: {:.2f}%".format(accuracy * 100))

# Print important features
print("Important Features:")
for feature in selected_feature_names:
    print(feature)

accuracy_poly_train = svm_selected_poly.score(X_trainScaled[:, selected_features], y_train)
print("Accuracy with selected features: {:.2f}%".format(accuracy_rbf * 100))

# Compute permutation importance
result_poly = permutation_importance(svm_selected_poly, X_testScaled[:, selected_features],
y_test, n_repeats=10, random_state=1)

# Extract feature names
feature_names = selected_feature_names

# Create dataframe to store feature importance
importance_df = pd.DataFrame({'feature': feature_names, 'importance':
result_poly.importances_mean})

# Sort dataframe by importance
importance_df = importance_df.sort_values(by='importance',
ascending=False).reset_index(drop=True)

# Plotting
plt.figure(figsize=(10, 8))
sns.barplot(x='importance', y='feature', data=importance_df, palette='viridis')
```

```
plt.title('Feature Importance')
plt.xlabel('Importance Rate')
plt.ylabel('Feature')
plt.show()

X_poly = data[['COSTWATR', 'AGE','BEDROOMS', 'ROOMS', 'DENSITY', 'HHINCOME', 'COSTGAS', 'COSTELEC','BUILTYR2', 'EDUC']]
X_trainPoly, X_testPoly, y_train, y_test =  train_test_split(X_poly, y, train_size = 0.7, test_size = 0.3
                                    ,random_state = 1)

scaler = StandardScaler().fit(X_trainPoly)
X_trainPolyScaled = scaler.transform(X_trainPoly)
X_testPolyScaled = scaler.transform(X_testPoly)

svm_poly=SVC(kernel='poly', cache_size=1000, verbose=True, max_iter=10000,degree=3, C=1, random_state=42)
svm_poly.fit(X_trainPolyScaled, y_train)


kfold = skm.KFold(5,
            random_state=42,
            shuffle=True)
grid = skm.GridSearchCV(svm_poly,
                    {'C':[0.1,1,10,100,1000],
                     'degree':[2,3,4,5]},
                    refit=True,
                    cv=kfold,
                    scoring='accuracy')



grid.fit(X_trainPolyScaled, y_train)
grid.best_params_
grid.best_params_
grid.best_estimator_
best_params = grid.best_params_
best_accuracy = grid.best_score_ * 100

# Print the best parameters and best accuracy
print("Best Parameters:", best_params)
print("Best Accuracy: %.2f%%" % best_accuracy)

#Best Parameters: {'C': 1, 'degree': 3}
```

```
#Best Accuracy: 84.53%

# Extracting the grid search results
results = grid.cv_results_

# Extracting the mean test scores
mean_test_scores = results['mean_test_score']

# Reshaping the mean test scores to match the grid of C and degree values
mean_test_scores = np.array(mean_test_scores).reshape(len(c_values), len(degree_values))

# Plotting the heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(mean_test_scores, annot=True, cmap="YlGnBu", xticklabels=degree_values, yticklabels=c_values)
plt.title('Accuracy for different C and degree values')
plt.xlabel('Degree')
plt.ylabel('C')
plt.show()

poly_svc = SVC(kernel='poly', cache_size=1000, verbose=True, max_iter=10000,degree=3, C=1, random_state=42)

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

# Evaluate the model on the testing data and print the accuracy score
accuracy = poly_svc.score(X_testPolyScaled, y_test)
print("poly SVM with C=1 and degree=3 on test set : %.2f%%" % (accuracy * 100))
accuracy = poly_svc.score(X_trainPolyScaled, y_train)
print("poly SVM with C=1 and degree= 3 on train set: %.2f%%" % (accuracy * 100))
#the train and test is almost the same

y_hat = poly_svc.predict(X_testPolyScaled)
conf = ConfusionMatrixDisplay.from_predictions(y_test, y_hat)
conf

X_top_poly = data[['Bedroom', 'AGE']]


#spliting the data into train and test
X_trainTop_poly, X_testTop_poly, y_train, y_test =  train_test_split(X_top_poly, y, train_size = 0.7, test_size = 0.3
                                    ,random_state = 1)
```

```
scaler = StandardScaler().fit(X_trainTop_poly)
X_trainScaled_Top_poly = scaler.transform(X_trainTop_poly)
X_testScaled_Top_poly = scaler.transform(X_testTop_poly)

polysvc_top = SVC(kernel='poly', cache_size=1000, verbose=True, max_iter=10000,degree=3,
C=1, random_state=42)
polysvc_top.fit(X_trainScaled_Top_poly, y_train)

fig, ax = subplots(figsize=(8,8))
plot_svm(X_trainScaled_Top_poly,
        y_train,
        polysvc_top,
        ax=ax)
ax.set_xlabel('BEDROOMS')
ax.set_ylabel('AGE')
plt.title('SVM Decision Boundary with Top 2 Predictor Variables. 1- Owned, 2- Rented.')
# Add legend with colors
class_labels = ['OWNED', 'RENTED']

# Plot legend
plt.legend(class_labels, title='OWNERSHIP')

plt.show()

accuracy = polysvc_top.score(X_testScaled_Top_poly, y_test)
print("Polynomial SVM top two with degree=3, C=1 on test set : %.2f%%" % (accuracy * 100))
accuracy =polysvc_top.score(X_trainScaled_Top_poly, y_train)
print("Polynomial SVM top two with degree=3, C=100 on train set: %.2f%%" % (accuracy * 100))

fig, ax = subplots(figsize=(8,8))
for (X_, y_, c, name) in zip(
    (X_trainPolyScaled, X_testPolyScaled),
    (y_train, y_test),
    ('r', 'b'),
    ('CV tuned on training $\gamma=0.5$',
     'CV tuned on test $\gamma=0.5$')):
   roc_curve(poly_svc,
        X_,
        y_,
        name= name,
        color=c,
        ax=ax)
fig;
```

```
def plot_decision_regions(X, y, classifier, resolution=0.02):
    # setup marker generator and color map
    markers = ('s', 'x', 'o', '^', 'v')
    colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan')
    cmap = plt.cm.RdYlBu

    # plot the decision surface
    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),
                           np.arange(x2_min, x2_max, resolution))
    Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
    Z = Z.reshape(xx1.shape)
    plt.contourf(xx1, xx2, Z, alpha=0.3, cmap=cmap)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())

    # plot class samples
    for idx, cl in enumerate(np.unique(y)):
        plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1],
                    alpha=0.8, c=[cmap(idx)],
                    marker=markers[idx], label=cl)

# Example usage:
# Assuming X_train and y_train are your training data
# and classifier is your trained classifier (e.g., SVM model)

plot_decision_regions(X_trainScaled_Top_poly, y_train, polysvc_top)
plt.xlabel('COSTWATR')
plt.ylabel('AGE')
plt.legend(loc='upper left')
plt.show()
```