REPORT: FACTORS AFFECTING HOUSE OWNERSHIP RATES

ABSTRACT

In this report, we delve into the application of support vector models (SVM) in predicting whether a dwelling is occupied by owners or renters. We achieve this by leveraging demographic and housing factors, which are vital indicators of occupancy status. Our analysis uses data collected via the US Census and accessed through IPUMS USA, which is a vast repository of information on people and their housing situations. We explore the relationships among five key features such as - age, income, education level, cost of electricity, and marital status - and use SVM with three different kernels (linear, radial, and polynomial) to model the relationship between these variables and occupancy status. Our findings demonstrate the predictive power of SVM in this context, with all three SVM models achieving high accuracy in predicting occupancy status. Moreover, our analysis reveals several interesting patterns within the data and those which seem to be strong predictors of homeownership.

INTRODUCTION

The study of housing problems and ownership status is critical in understanding the socioeconomic conditions common in any particular region. In this paper, we investigated the task of investigating the support vector models' (SVM) capacity to predict whether a housing unit is owned or rented, depending on housing and demographic factors.

The primary goals are to learn more about SVM's prediction capabilities and find intriguing patterns in the data. With the help of this analysis, we seek to pinpoint the strong predictors that substantially impact the occupancy status of housing units and showcase how these factors could possibly be utilized to produce accurate predictions.

And stakeholders and policymakers can arrive at better judgments about housing regulations and offer greater support to people and families looking for cheap and sustainable housing options by understanding the elements that are important in determining occupancy status. Our research could be utilized as a tool to improve understanding of ownership status and housing conditions in general and to promote the creation of policies that effectively address housing challenges in the area.

THEORETICAL BACKGROUND

Support Vector Machines (SVM) is a popular supervised machine learning algorithm that is widely used for classification and regression tasks. SVM is based on the concept of finding a hyperplane that separates the data points into different classes while maximizing the margin between the hyperplane and the closest data points. SVM can use different types of kernels, such as linear, radial, and polynomial, to transform the data into a higher-dimensional space, making it easier to find a separating hyperplane. Each kernel has its own unique features and is suitable for different types of data.

The linear kernel is the simplest type of kernel and is commonly used for linearly separable datasets. It maps the data into a higher-dimensional space by using a linear function. The linear kernel is efficient and easy to use, but it may not be suitable for datasets that are not linearly separable. And radial kernel, also known as the Gaussian kernel, is commonly used for non-linearly separable datasets. It maps the data into a higher-dimensional space using a non-linear function that creates a circular decision boundary. The radial kernel is more flexible than the linear kernel and can handle complex datasets, but it is computationally expensive. And also there is polynomial kernel is used to transform the data into a higher-dimensional space using a polynomial function. It is suitable for datasets that have non-linear relationships between the features. The polynomial kernel is computationally less expensive than the radial kernel, but it can be sensitive to the choice of the polynomial degree.

The choice of kernel depends on the nature of the dataset and the complexity of the decision boundary. In addition to these kernels, there are other types of kernels such as sigmoid, and ANOVA kernels, which are used in specific applications. SVM has been successfully applied to a wide range of applications, including image recognition, natural language processing, and bioinformatics. SVM's ability to handle high-dimensional data and its strong generalization ability make it a popular choice for many classification and regression problems.

TECHNICAL BACKGROUND

Support Vector Machines (SVMs) is a powerful supervised machine learning algorithms that can be used for both classification and regression problems. When the number of features is large and the number of data points is low, it is especially helpful. The foundation of SVMs is the idea of locating the hyperplane in the feature space that best divides the data into various groups.

If the data cannot be separated linearly, we can translate the data into a higher-dimensional space using a kernel function so that it can be separated linearly. The regularization parameter (C), the kernel function, the kernel parameter (gamma), and other hyper parameters of SVMs might influence how well the technique performs. Parameter tuning is the process of determining the ideal hyper parameters for an SVM.

Any machine learning project must include parameter tuning, which entails choosing the hyper parameters' ideal values in order to maximize the model's performance. One common technique for parameter tuning is Grid Search, which involves training and evaluating the SVM model for a range of hyper parameter values, and selecting the combination of values that yields the best performance on a validation set.

Two of the most crucial SVM parameters, the cost and gamma parameters, can significantly affect the model's performance. The trade-off between attaining a low training error and a low testing error is controlled by the cost parameter, also referred to as the C parameter. It establishes the punishment for incorrectly classifying data points in the training set. The model should penalize misclassification more harshly if C is higher, which would lead to a smaller margin and more support vectors. A lower value of C, on the other hand, leads to a greater margin and fewer support vectors.

A broader margin will result from a lower value of C, which can lessen the chance of over fitting the training set. However, it may also increase the chance of under fitting the data, which means the model will not generalize well to new data. A greater value of C, on the other hand, will result in a narrower margin and could cause the data to be over fitted, which would mean that the model would perform well on the training data but poorly on the test data.

The gamma parameter regulates the decision boundary's shape and can greatly affect the effectiveness of the model. The decision boundary becomes more complex and is more prone to overfit the data when gamma is larger. The decision boundary becomes smoother and is more likely to underfit the data when gamma is lower.

Generally speaking, determining the best values for the cost and gamma parameters is a crucial stage in creating an SVM model and necessitates some trial and error. To find the ideal values for these parameters, cross-validation is frequently utilized. By carefully adjusting the cost and gamma parameters, it is possible to establish the ideal balance between bias and variance in the model.

METHODOLOGY:

To start our analysis of the housing situation and ownership status, we begin by exploring the data with variables, namely age, income, education level, cost of electricity, and marital status whether they are married or divorced or not married , to investigate their relationship with occupancy status. We preprocessed the data by normalizing the features and splitting it into training and test sets. We also used cross-validation to determine the appropriate sample size from the dataset of almost 75000 records and tuned several cost values to select the ideal cost value.

We made some data transformations to improve the performance of the SVM models. Specifically, we created dummy variables for the marital status column, dividing it into several categorical columns such as status married, status divorced, status single, etc. Additionally, we filtered our dataset to remove redundancy, as we found that there were multiple persons registered with the same serial number. To address this, we chose the person with the oldest age in a given dwelling to be our selection instead of modeling all persons in a single home.

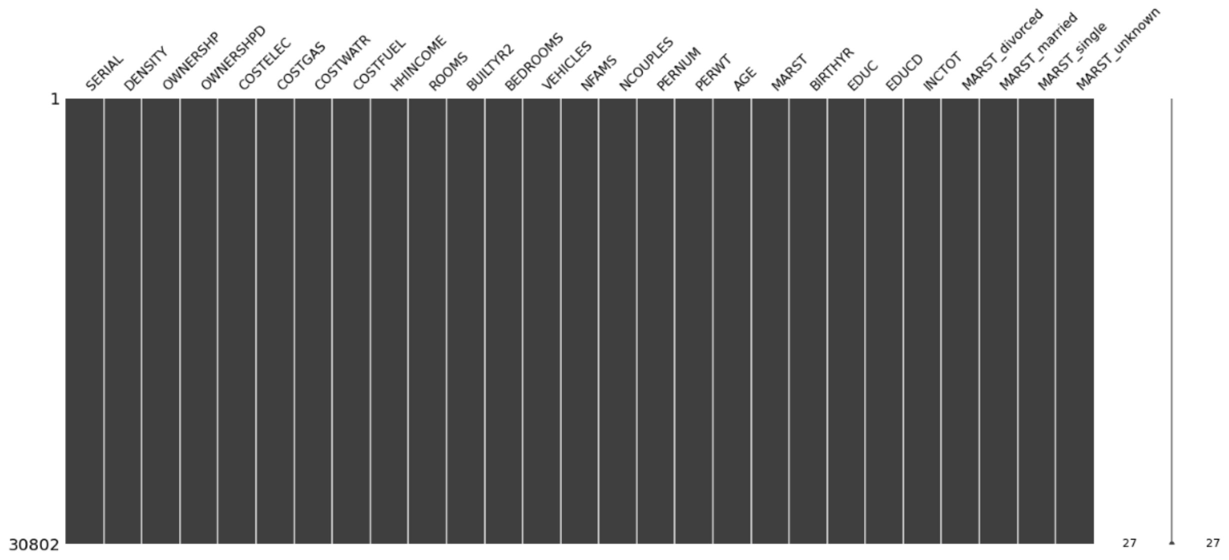


Fig 4.1: Cleaned Dataset after Transformation.

And above given missing values plot shows that our dataset is clean and there are about 30000 records after all the transformation on data has been performed. These data transformations would be essential in improving the accuracy of our models, as they help to eliminate noise and focus on the most relevant variables. We trained three SVM models using linear, radial, and polynomial kernels and tuned the hyper parameters using cross-validation. We evaluated the models' performance using accuracy, precision.. By analyzing the models' results, we identified the key factors that influenced occupancy status and how these factors could be leveraged to make accurate predictions. Thus, we will be able to generate a comprehensive analysis of the factors that influence occupancy status and develop a robust predictive model that accurately classified dwellings into owner-occupied or rental categories.

In addition to, we explored several other variables such as marital status electricity cost, water cost, gas cost, population density, age of the person living in the home, and household income. We fit the SVM between any two features and described the relationships between them. To achieve this, we used different kernels, including linear, radial, and polynomial kernels. The linear kernel was useful in cases where the data was separable with a straight line, and the radial kernel was used when the decision boundary was not a straight line. The polynomial kernel was useful when the data was not linearly separable, and it mapped the data into a higher-dimensional space, making it easier to find a separating hyperplane.

COMPUTATIONAL RESULTS:

We will now look at a number of plots to determine which variables are most effective at predicting housing status. By doing this, we expect to learn more about the important factors that our predictive model should take into account.

We analyze the relationship between various variables and occupancy status through the following charts. We can decide which variables to include in our model by examining these plots to look for any patterns or trends that may be present in the data.

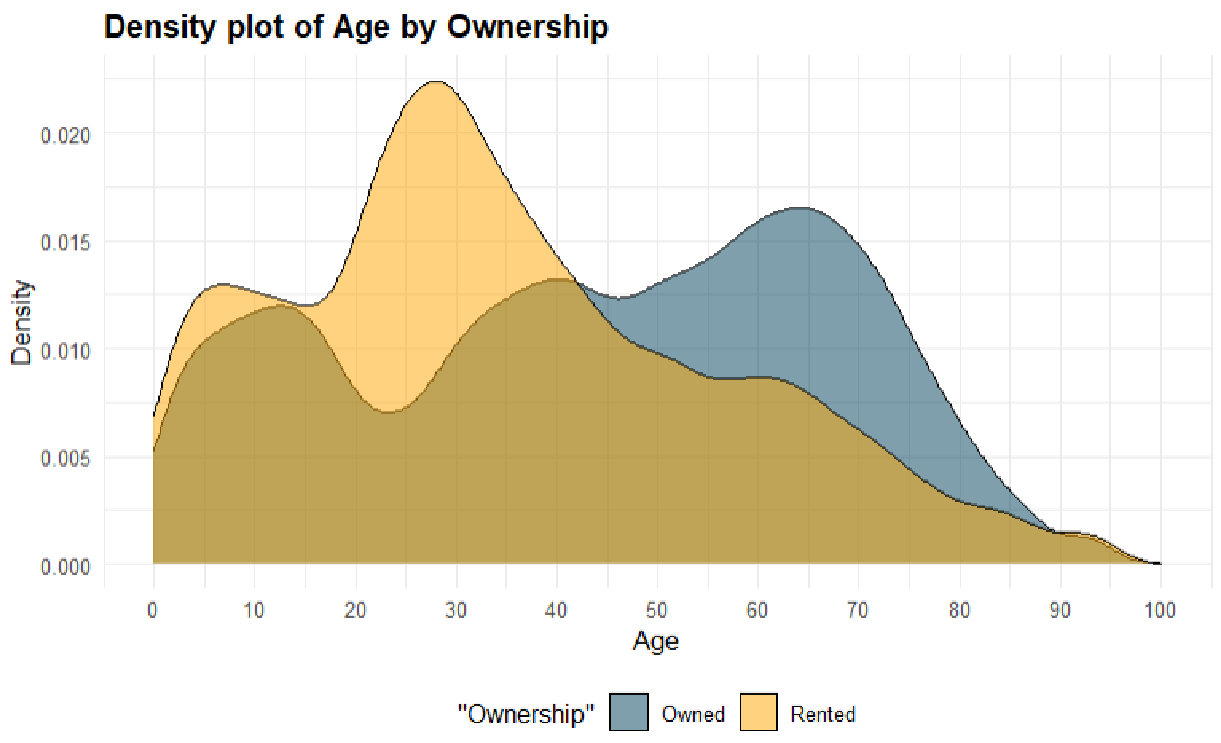


Fig 5.1: Density plot

The graph we've shown suggests a possible relationship between population density and homeownership rates. Particularly, the likelihood that residents of a given area will own a home rises as the population density of that area does. In contrast, as population density declines, it becomes more likely that residents will rent out their homes rather than own them.

When attempting to estimate the housing status of a certain place, this relationship between population density and homeownership rates is an important factor to take into account. By examining this pattern, we may learn important lessons about the factors that influence house ownership rates from the ground up and utilize this knowledge to develop more precise housing market forecasts

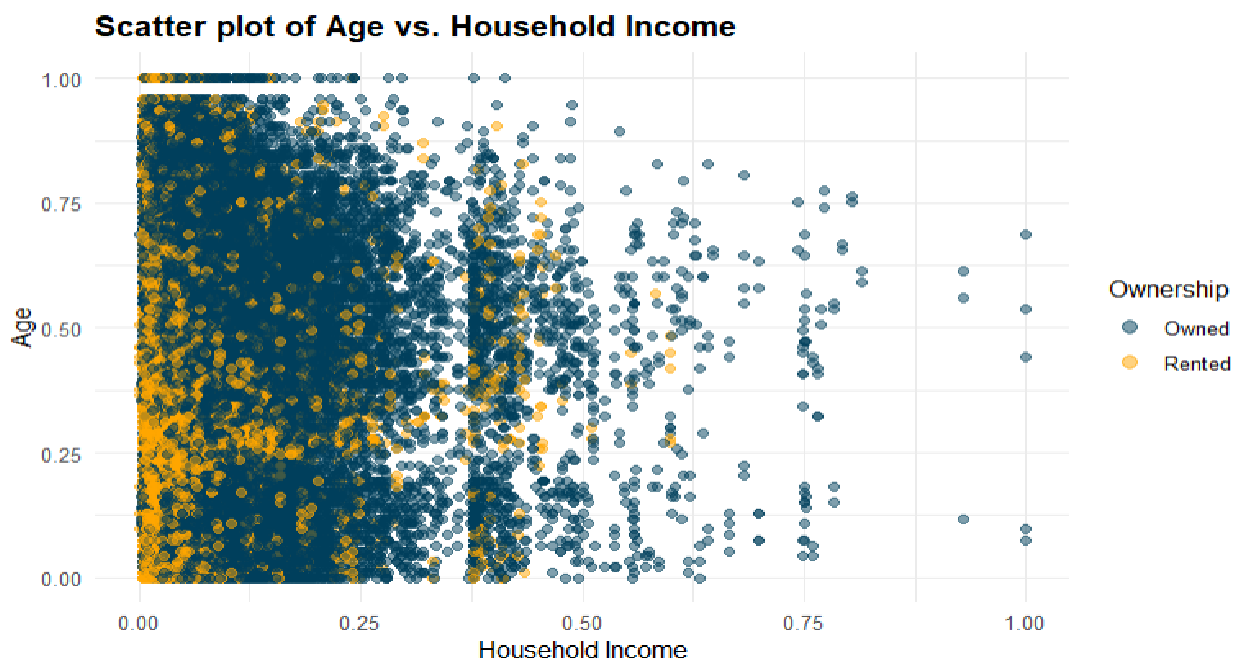


Fig 5.2: Scatter Plot

It is clear from the previously displayed visual that household income and home ownership status are related. More specifically, the likelihood that residents own a home in the neighborhood rises when household income rises above a cutoff of $24,000. In contrast, as household income falls below a cutoff of $12,000, there is a greater likelihood that the local residences are rented as opposed to owned: this finding implies that household income is one of the key factors in deciding whether a person is a homeowner and that it can be used as a predictor when creating a model to divide homes into owner-occupied and rented categories.

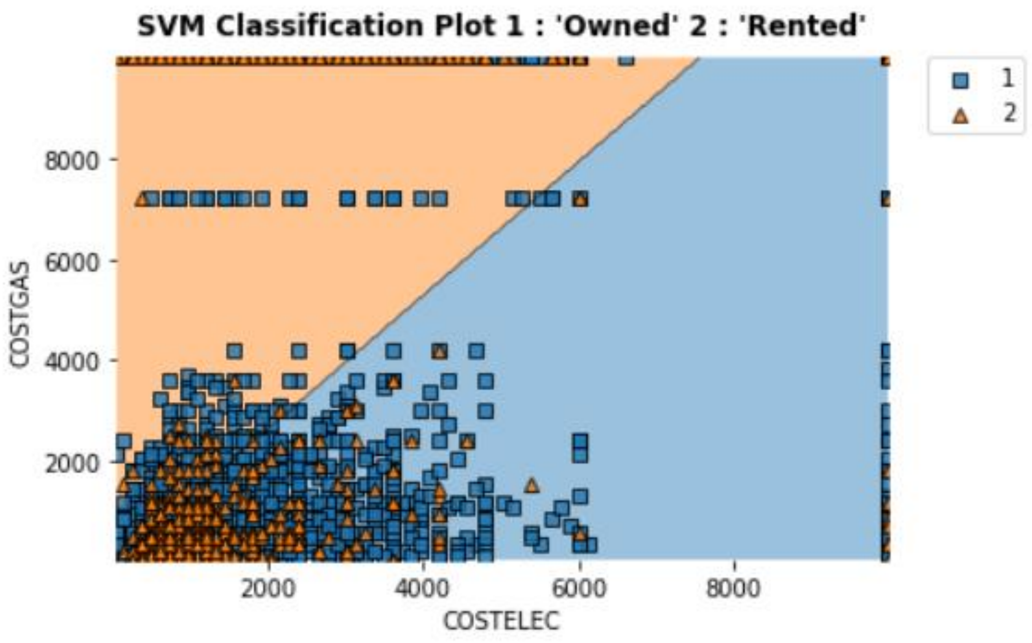


Fig 5.3: Linear SVM classification between Electricity and Gas prices

After that, we looked at the SVM model's decision boundary using the polynomial kernel and discovered that it successfully divided the data into two classes that represented either owned or rented residences. We then used Radial Kernel for our data but were unable to view the decision boundary but instead we were able to get high test and training accuracy for our model and can be viewed in Fig 5.4 below.

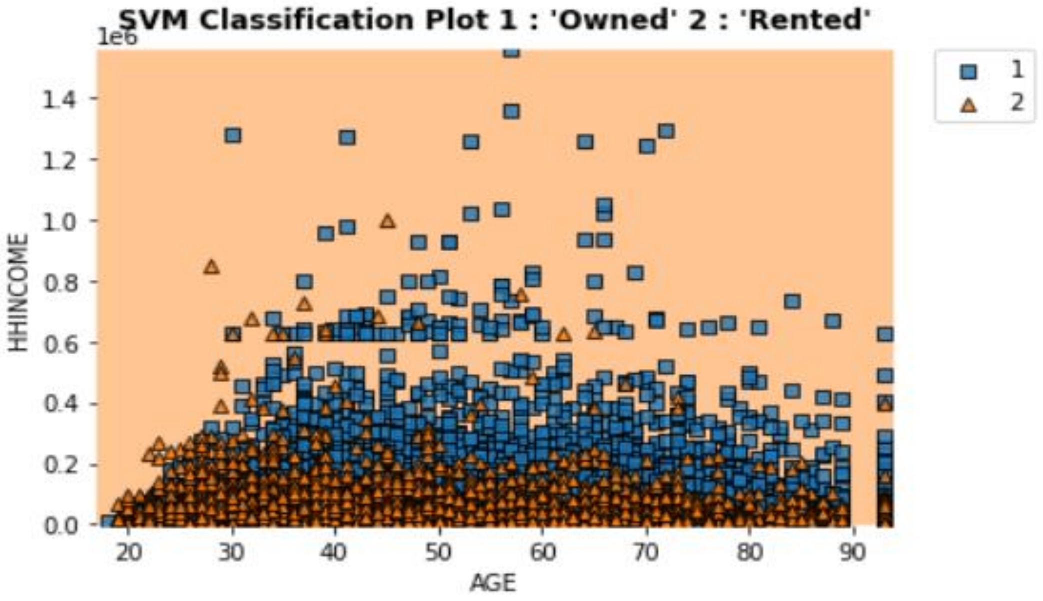


Fig 5.4: Radial Kernel for Age versus Household Income

After analysis, we found that the degree 4 polynomial curve offered a better match for the data, according to our analysis of the data. It was interesting to learn that the linear kernel also predicted the ownership categorical value with a high degree of accuracy.

According to the plot below (Fig 5.5), those who live in joint households with six or more family members own more property. The graph shows how the size of the family and their marital status are directly related. Larger families may have greater income levels and multiple sources of income available to purchase a home, which may account for this. Additionally, it is typical for extended families to live together in some cultures, and having a home of one's own can give the entire family a feeling of security and stability.

It is important to remember that while the plot suggests that joint families have a greater ownership percentage, this does not imply that all joint families are homeowners. Whether someone owns or rents a property depends on a variety of other circumstances, including their income, marital status, and where they live. The plot does, however, imply that joint families might have an edge when it comes to homeownership, and stakeholders and policymakers may want to take this information into account when creating housing laws or initiatives.

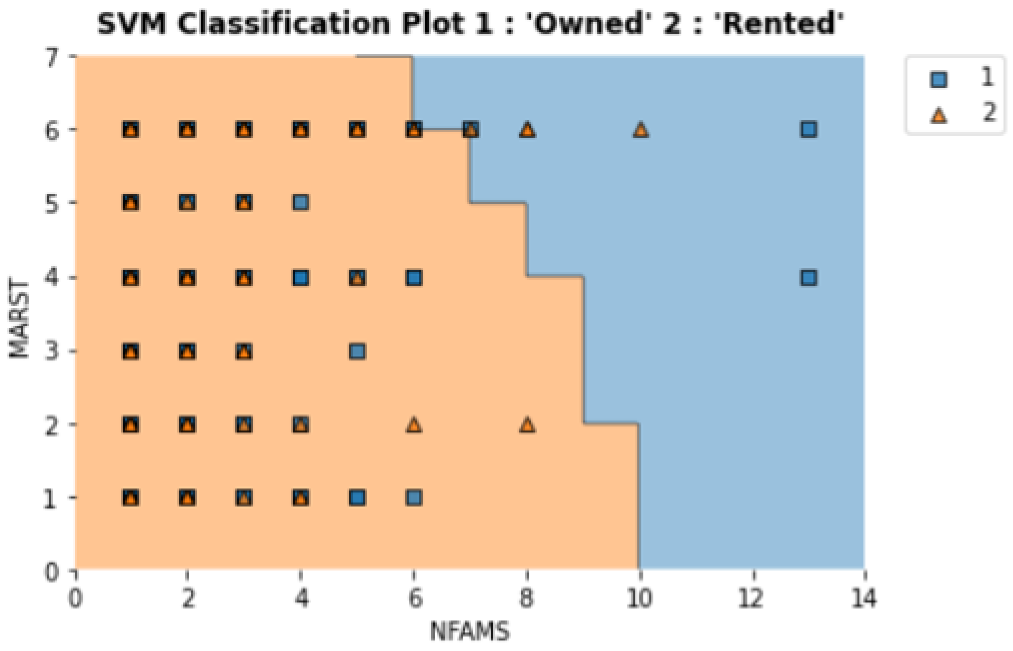


Fig 5.5: Polynomial SVM for Num. of Families vs. Marital Status

The polynomial curve showed a decision boundary that depicted the correlation between the chosen variables. However, the straight line from the linear kernel performed surprisingly well in predicting ownership status.

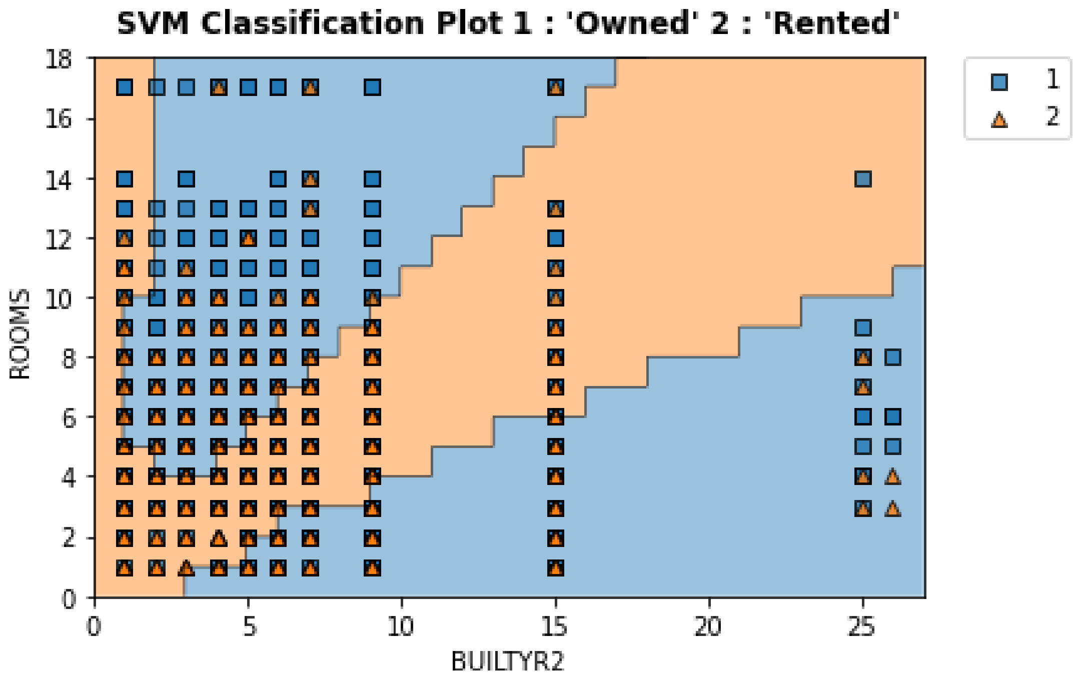


Fig 5.6: Early Room V.s Built Year (Poly Kernel)

Overall, we can draw the conclusion that the type of kernel to use depends on the data and how the variables are related. The linear kernel can occasionally be surprisingly successful, even though the polynomial kernel is useful for non-linearly separable data. To ascertain, it is crucial to investigate various kernels.

And we now try to compare all our kernels across training and testing, to check either of training set or testing set performed better in our analysis:

BASIS FUNCTION	TRAINING SET	TEST SET	BEST PARAMETERS
Linear Kernel	Moderate	Good	Cost: 27, Sample Size:15000
Radial Kernel	Good	Good	Cost: 27, Gamma:1
Polynomial Kernel	Poor	Best	Cost:12, Degree:4

And we try plotting our decision boundaries across severable variables and found out that the following will give best results

BASIS FUNCTION	TRAINING_ACCURACY	TESTING_ACCURACY	STRONGEST_PREDICTORS
Linear Kernel	71.15%	70.32%	No linear relationship between any factors.
Radial Kernel	71.17%	78.64%	[‘AGE’,’HHICOME’]
Polynomial Kernel	51.53%	73.28%	[‘BUILTYR,’ROOMS’], ['COSTELEC','COSTGAS']

Overall, “**Radial Kernel**” was performing better performing for both training and testing sets in determining factors predicting the ownership rate across a region. And as Linear Kernel was performing better at test set it was performing bad as there were no linear relationships existing in the data. Conversely, polynomial was bad at training set but it was performing best at the test set but as we increase the degree of the polynomial it tends to uncover hidden pattern, but in due course of finding the trend it is over fitting the data in our dataset leading to less accuracy and high training and test error rates.

DISCUSSION:

The analysis explores the use of support vector models (SVM) in predicting whether a dwelling is owned or rented, based on various demographic and housing factors. The aim of the analysis is to identify the strong predictors that significantly impact the occupancy status of housing units. The data is collected through the US Census, which includes information on people and their housing situations, and accessed through IPUMS USA.

In the analysis, we investigated the relationship between the key variables or factors (age of the person, household income, population density, cost of electricity and marital status) and occupancy status. Then the data is preprocessed by normalizing the features and splitting it into training and test sets. Furthermore, some data transformations have been applied to improve the performance of the SVM models. Specifically, creating dummy variables for the marital status column and filtering the dataset aided in controlling data redundancy and meaningful results.

By analyzing the models' results, the key factors that influenced occupancy status and how these factors could be leveraged to make accurate predictions were discussed and explored in detail. The study identified that people who have higher total income and are not married are likely to own a home rather than renting the home.

Moreover, people who are in a house with 6 family counts and married are likely to have their own home, and those who are in a 2 family count are likely to rent a home. The best performing model in the analysis was SVM model with radial kernel, and next up was polynomial kernel with degree 4 for more accurate predictions.

However, it should be noted that there is a potential ethical concern with using demographic and housing factors to predict whether a dwelling is owned or rented. This type of analysis could perpetuate systemic inequalities, such as discrimination against marginalized groups, and lead to exclusionary practices in housing.

Therefore, it is crucial to consider the ethical implications of using such models and ensure that they do not result in discriminatory practices. The research findings could help stakeholders and policymakers make better decisions about housing regulations and offer greater support to people and families looking for affordable and sustainable housing options while also ensuring ethical considerations are taken into account.

<u>**CONCLUSION:**</u>

Our analysis has revealed that a number of important variables are critically important in determining a home's ownership rate. Age, household income, power costs, and marital status, particularly being single, were found to be highly predictive of ownership status, according to our analysis. Renting a property is more common among those whose total income is higher and who are single. Based on our findings, we recommend that policymakers and stakeholders pay attention to these issues in order to make educated decisions about housing rules and to assist families looking for cheap and sustainable housing solutions.

The SVM model with radial kernel turned out to be the best-performing model in our investigation, followed by the polynomial kernel with degree 4 for more precise predictions. Additionally, those who live in high-density areas and have monthly gas and electricity bills of $4,000 or less are more likely to rent a home, whereas those with higher bills are more likely to own the home.

All in all, our analysis offers insightful information on the variables affecting ownership status and might be a useful tool for stakeholders and policymakers looking to promote sustainable and affordable housing options.

<u>**REFERENCES:**</u>

IPUMS USA. dataset (n.d.). Retrieved from https://usa.ipums.org/usa/index.shtml

APPENDIX:

In [54]:
```
import pyreadr
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, plot_confusion_matrix
import seaborn as sns
from sklearn.model_selection import cross_val_score
# standard scaler code
from sklearn.preprocessing import StandardScaler
import warnings
from sklearn.model_selection import GridSearchCV
warnings.filterwarnings("ignore")
from mlxtend.plotting import plot_decision_regions
from sklearn.model_selection import train_test_split, GridSearchCV
from collections import Counter
import missingno as msno
```

In [55]:
```
result = pyreadr.read_r("C:/Users/GOKUL/Downloads/Housing.Rdata")
```

In [56]:
```
print(result.keys())
```

```
odict_keys(['data'])
```

In [57]:
```
data = result["data"]
```

In [58]:
```
data
```

Out[58]:

	SERIAL	DENSITY	OWNERSHP	OWNERSHPD	COSTELEC	COSTGAS	COSTWATR	COSTF
0	1371772.0	920.0	1	13	9990.0	9993.0	360.0	99
1	1371773.0	3640.9	2	22	1080.0	9993.0	1800.0	99
2	1371773.0	3640.9	2	22	1080.0	9993.0	1800.0	99
3	1371774.0	22.5	1	13	600.0	9993.0	9993.0	99
4	1371775.0	3710.4	2	22	3600.0	9993.0	9997.0	99
...	...	...	...	...	...	...	...	
75383	1402573.0	2754.9	2	22	9990.0	7200.0	960.0	99
75384	1402573.0	2754.9	2	22	9990.0	7200.0	960.0	99
75385	1402573.0	2754.9	2	22	9990.0	7200.0	960.0	99
75386	1402573.0	2754.9	2	22	9990.0	7200.0	960.0	99
75387	1402573.0	2754.9	2	22	9990.0	7200.0	960.0	99

75388 rows × 23 columns

In [59]:
```
# create a dictionary to map values to labels
, 4: 'divorced', 6: 'single', 3: 'unknown', 5:
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
# use the replace function to map values to labels, and replace all other values w
#data['MARST'] = data['MARST'].replace(labels)
data['MARST'] = data['MARST'].replace(labels)
```

In [60]:
```
Counter(data['MARST'])
```

Out[60]:
```
Counter({'single': 29892, 'divorced': 7146, 'unknown': 3864, 'married': 34486})
```

In [61]:
```
# we use get_dummies to create a new dataframe with one column for each unique valu
dummies = pd.get_dummies(data['MARST'], prefix='MARST')

# concatenate the original dataframe with the new dummy variable columns
data = pd.concat([data, dummies], axis=1)

data = data.drop('MARST', axis=1)
```

In [62]:
```
# sort the dataframe by 'SERIAL' and 'AGE' in descending order
data = data.sort_values(['SERIAL', 'AGE'], ascending=[True, False])

# drop duplicates based on 'SERIAL', keeping the first occurrence
data = data.drop_duplicates(subset='SERIAL', keep='first')

# reset index starting from 1
data = data.reset_index(drop=True)
data.index += 1

print(data)
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
          SERIAL  DENSITY  OWNERSHP  OWNERSHPD  COSTELEC  COSTGAS  COSTWATR  \
1      1371772.0    920.0         1         13    9990.0   9993.0     360.0
2      1371773.0   3640.9         2         22    1080.0   9993.0    1800.0
3      1371774.0     22.5         1         13     600.0   9993.0    9993.0
4      1371775.0   3710.4         2         22    3600.0   9993.0    9997.0
5      1371776.0    448.2         1         12    1560.0   3000.0    9993.0
...          ...      ...       ...        ...       ...      ...       ...
30798  1402569.0    362.4         1         13    1200.0   9993.0     500.0
30799  1402570.0    667.7         1         12     600.0   1200.0     750.0
30800  1402571.0   2425.1         1         13    2400.0    600.0    1400.0
30801  1402572.0   2295.9         1         13    1200.0   9992.0    1600.0
30802  1402573.0   2754.9         2         22    9990.0   7200.0     960.0

       COSTFUEL  HHINCOME  ROOMS  ...  PERWT  AGE  BIRTHYR  EDUC  EDUCD  \
1        9993.0   75000.0      7  ...   14.0   52   1969.0     7     71
2        9993.0   13600.0      6  ...   83.0   22   1999.0    10    101
3        9993.0    7000.0      5  ...   33.0   62   1959.0     6     63
4        9993.0   50500.0      4  ...  297.0   50   1971.0     7     71
5        9993.0  155300.0      5  ...   10.0   93   1928.0    10    101
...         ...       ...    ...  ...    ...  ...      ...   ...    ...
30798    9993.0  106000.0      7  ...  131.0   51   1970.0     6     63
30799    9993.0  121500.0      6  ...  115.0   65   1956.0    11    114
30800    9993.0  201800.0      8  ...   63.0   70   1951.0    11    114
30801    9993.0   75500.0      6  ...  172.0   70   1951.0     6     63
30802    9993.0   86700.0      8  ...  157.0   64   1957.0     6     63

         INCTOT  MARST_divorced  MARST_married  MARST_single  MARST_unknown
1       75000.0               0              0             1              0
2        5600.0               0              0             1              0
3        7000.0               1              0             0              0
4       16000.0               0              0             0              1
5       89800.0               0              1             0              0
...         ...             ...            ...           ...            ...
30798   20000.0               1              0             0              0
30799  100000.0               0              1             0              0
30800  149000.0               0              1             0              0
30801    9600.0               0              1             0              0
30802    1700.0               0              1             0              0

[30802 rows x 26 columns]
```

In [63]:
```
data.columns
```

Out[63]:
```
Index(['SERIAL', 'DENSITY', 'OWNERSHP', 'OWNERSHPD', 'COSTELEC', 'COSTGAS',
       'COSTWATR', 'COSTFUEL', 'HHINCOME', 'ROOMS', 'BUILTYR2', 'BEDROOMS',
       'VEHICLES', 'NFAMS', 'NCOUPLES', 'PERNUM', 'PERWT', 'AGE', 'BIRTHYR',
       'EDUC', 'EDUCD', 'INCTOT', 'MARST_divorced', 'MARST_married',
       'MARST_single', 'MARST_unknown'],
      dtype='object')
```

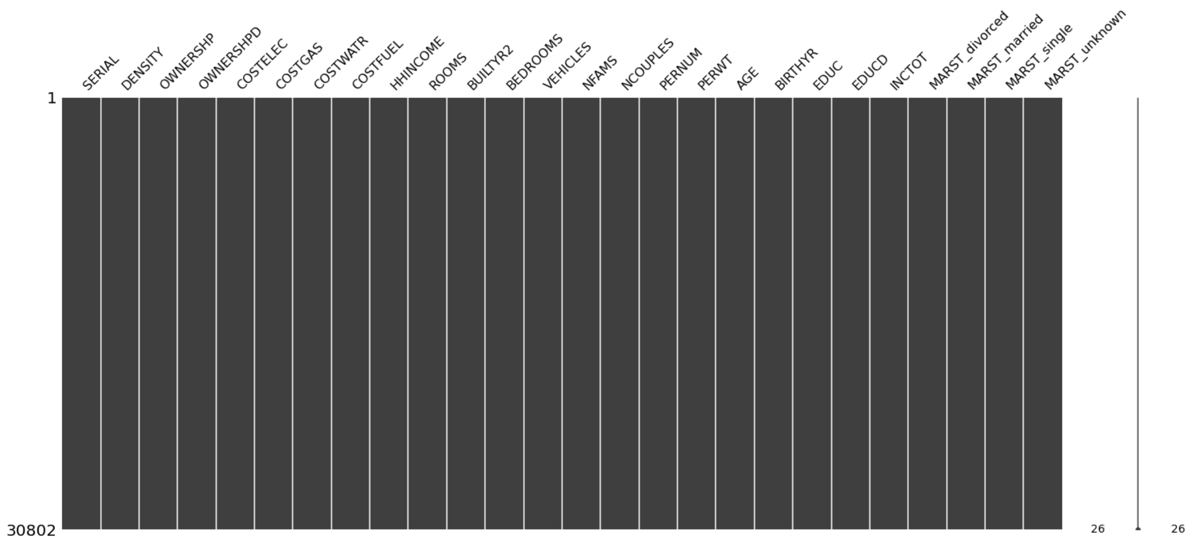
In [64]:
```
data.info()
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30802 entries, 1 to 30802
Data columns (total 26 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   SERIAL          30802 non-null  float64
 1   DENSITY         30802 non-null  float64
 2   OWNERSHP        30802 non-null  int32  
 3   OWNERSHPD       30802 non-null  int32  
 4   COSTELEC        30802 non-null  float64
 5   COSTGAS         30802 non-null  float64
 6   COSTWATR        30802 non-null  float64
 7   COSTFUEL        30802 non-null  float64
 8   HHINCOME        30802 non-null  float64
 9   ROOMS           30802 non-null  int32  
 10  BUILTYR2        30802 non-null  int32  
 11  BEDROOMS        30802 non-null  int32  
 12  VEHICLES        30802 non-null  int32  
 13  NFAMS           30802 non-null  int32  
 14  NCOUPLES        30802 non-null  int32  
 15  PERNUM          30802 non-null  float64
 16  PERWT           30802 non-null  float64
 17  AGE             30802 non-null  int32  
 18  BIRTHYR         30802 non-null  float64
 19  EDUC            30802 non-null  int32  
 20  EDUCD           30802 non-null  int32  
 21  INCTOT          30802 non-null  float64
 22  MARST_divorced  30802 non-null  uint8  
 23  MARST_married   30802 non-null  uint8  
 24  MARST_single    30802 non-null  uint8  
 25  MARST_unknown   30802 non-null  uint8  
dtypes: float64(11), int32(11), uint8(4)
memory usage: 4.0 MB
```

In [65]:
```
msno.matrix(data)
```

Out[65]:
```
<AxesSubplot:>
```



In [ ]:

"Appropriate Sampling of a large data set for linear kernel"

In [66]:
```
# list of sample sizes to try
sample_sizes = [5000, 10000, 15000, 20000]
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
# dictionary to store the cross-validation scores
scores = {}

# loop over sample sizes
for size in sample_sizes:
    # sample the data
    sampled_data = data.sample(n=size, replace=False)

    # train-test split
    data_train, data_test = train_test_split(sampled_data, test_size=0.5, random_s

    # fit the model
    svmfit_1 = SVC(kernel='linear', C=10, cache_size=1000, verbose=True, max_iter=
    svmfit_1.fit(data_train.drop('OWNERSHP', axis=1), data_train['OWNERSHP'])

    # perform cross-validation
    cv_scores = cross_val_score(svmfit_1, np.array(sampled_data.drop('OWNERSHP', a
                                np.array(sampled_data['OWNERSHP']), cv=5)

    # store the mean score
    scores[size] = cv_scores.mean()
# select the sample size with the highest cross-validation score
best_size = max(scores, key=scores.get)

print(scores.items())

# sample the data with the best size
sampled_data = data.sample(n=best_size, replace=False)
```

```
[LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][L
ibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][Lib
SVM][LibSVM][LibSVM][LibSVM]dict_items([(5000, 0.6534), (10000, 0.6747), (15000,
0.5949333333333333), (20000, 0.5989500000000001)])
```

In [67]:
```
# train-test split
data_train, data_test = train_test_split(sampled_data, test_size=0.5, random_state
```

lets check for various cost values now and check out the optimal cost over a subset of 'best_size' rows.

In [80]:
```
# Train SVM with linear kernel for various cost values
cost_values = [i for i in range(10,200,2)]
train_error = []
test_error = []
for c in cost_values:
    svmfit = SVC(kernel='linear', C=c, cache_size=1000, verbose=True, max_iter=100
    svmfit.fit(data_train.drop('OWNERSHP', axis=1), data_train['OWNERSHP'])

    # Make predictions on training set
    y_train_pred = svmfit.predict(data_train.drop('OWNERSHP', axis=1))
    train_accuracy = accuracy_score(data_train['OWNERSHP'], y_train_pred)
    train_error.append(1 - train_accuracy)

    # Make predictions on test set
    y_test_pred = svmfit.predict(data_test.drop('OWNERSHP', axis=1))
    test_accuracy = accuracy_score(data_test['OWNERSHP'], y_test_pred)
    test_error.append(1 - test_accuracy)

#     print("Cost:", c)
#     print("Training Error:", train_error[-1])
#     print("Test Error:", test_error[-1])
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
[LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][L
ibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][Lib
SVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSV
M][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM]
[LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][L
ibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][Lib
SVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSV
M][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM]
[LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][L
ibSVM][LibSVM][LibSVM]
```

In [81]:
```
min_cost = test_error.index(min(test_error))
print("Our minimal cost values is at C =",min_cost)
```

```
Our minimal cost values is at C = 80
```

In [77]:
```
#test_error[50:90]
```

And we can see that a cost value of 80 will give us a good result(least posiible error).

Linear Kernel:

In [82]:
```
# Convert OWNERSHP to a factor variable
data_train['OWNERSHP'] = data_train['OWNERSHP'].astype('category')
# Convert OWNERSHP to a factor variable"
data_test['OWNERSHP'] = data_test['OWNERSHP'].astype('category')
```

In [83]:
```
# Train SVM with linear kernel
svmfit_1 = SVC(kernel='linear', C=min_cost, cache_size=1000, verbose=True, max_ite
svmfit_1.fit(data_train.drop('OWNERSHP', axis=1), data_train['OWNERSHP'])


# Print summary of SVM model
print(svmfit_1)
```

```
[LibSVM]SVC(C=80, cache_size=1000, kernel='linear', max_iter=10000, random_state=4
2,
    verbose=True)
```

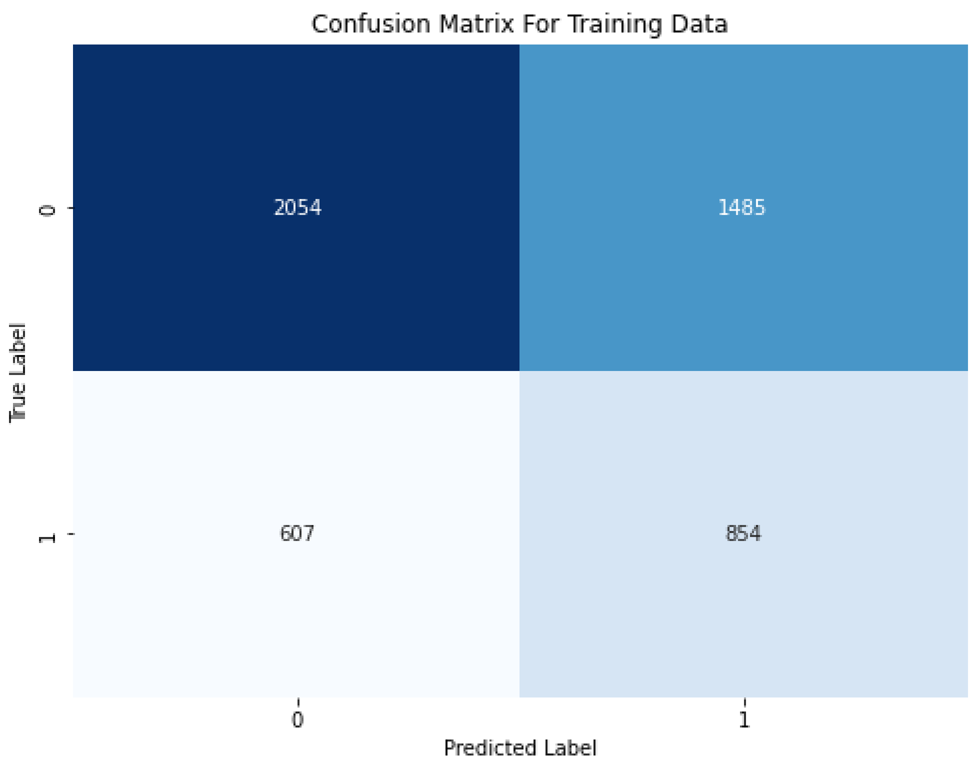
In [84]:
```
# Make predictions on training set
y_train_pred = svmfit_1.predict(data_train.drop('OWNERSHP', axis=1))

# Calculate training error and accuracy
train_error = 1 - accuracy_score(data_train['OWNERSHP'], y_train_pred)
train_accuracy = accuracy_score(data_train['OWNERSHP'], y_train_pred)

print("Training Error:", train_error)
print("Training Accuracy:", train_accuracy)
```

```
Training Error: 0.4184
Training Accuracy: 0.5816
```

In [85]:
```
# Make predictions on test set
y_test_pred = svmfit_1.predict(data_test.drop('OWNERSHP', axis=1))

# Calculate training error and accuracy
test_error = 1 - accuracy_score(data_test['OWNERSHP'], y_test_pred)
test_accuracy = accuracy_score(data_test['OWNERSHP'], y_test_pred)

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js
print("Test Accuracy:", test_accuracy)
```

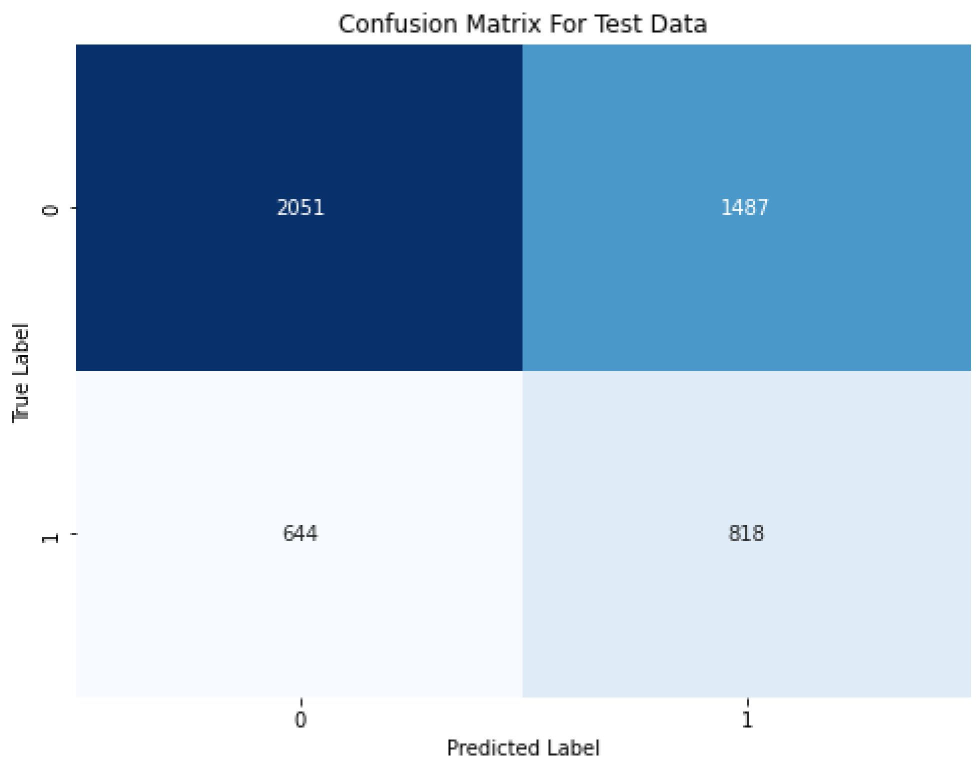
```
Test Error: 0.4262
Test Accuracy: 0.5738
```

In [86]:
```
# Calculate confusion matrix
cm = confusion_matrix(data_train['OWNERSHP'], y_train_pred)

# Plot confusion matrix
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.title('Confusion Matrix For Training Data')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```



In [87]:
```
# Calculate confusion matrix
cm = confusion_matrix(data_test['OWNERSHP'], y_test_pred)

# Plot confusion matrix
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.title('Confusion Matrix For Test Data')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



```
In [88]: sampled_data.columns
```

```
Out[88]: Index(['SERIAL', 'DENSITY', 'OWNERSHP', 'OWNERSHPD', 'COSTELEC', 'COSTGAS',
               'COSTWATR', 'COSTFUEL', 'HHINCOME', 'ROOMS', 'BUILTYR2', 'BEDROOMS',
               'VEHICLES', 'NFAMS', 'NCOUPLES', 'PERNUM', 'PERWT', 'AGE', 'BIRTHYR',
               'EDUC', 'EDUCD', 'INCTOT', 'MARST_divorced', 'MARST_married',
               'MARST_single', 'MARST_unknown'],
              dtype='object')
```

```
In [161…  def fit_two(v1,v2):
              # Convert OWNERSHP to a factor variable
              data_train['OWNERSHP'] = data_train['OWNERSHP'].astype('category')
              data_test['OWNERSHP'] = data_test['OWNERSHP'].astype('category')

              # Scale the data
              scaler = StandardScaler()
              data_train_scaled = scaler.fit_transform(data_train[[v1,v2]])

              # Train SVM with linear kernel
              svmfit_1 = SVC(kernel='linear', C=min_cost, cache_size=1000, verbose=True, max_
              svmfit_1.fit(data_train_scaled, data_train['OWNERSHP'])

              # Make predictions on test set
              data_test_scaled = scaler.fit_transform(data_test[[v1,v2]])
              y_test_pred = svmfit_1.predict(data_test_scaled)

              # Calculate training error and accuracy
              test_error = 1 - accuracy_score(data_test['OWNERSHP'], y_test_pred)
              test_accuracy = accuracy_score(data_test['OWNERSHP'], y_test_pred)

              print("Test Error:", test_error)
              print("Test Accuracy:", test_accuracy)
              # Make predictions on training set
              y_train_pred = svmfit_1.predict(data_train[[v1,v2]])

              # Calculate training error and accuracy
              train_error = 1 - accuracy_score(data_train['OWNERSHP'], y_train_pred)
                                                 (data_train['OWNERSHP'], y_train_pred)
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
    print("Training Error:", train_error)
    print("Training Accuracy:", train_accuracy)
    return svmfit_1
    #plot_two_variables(svmfit_1,v1,v2)
```

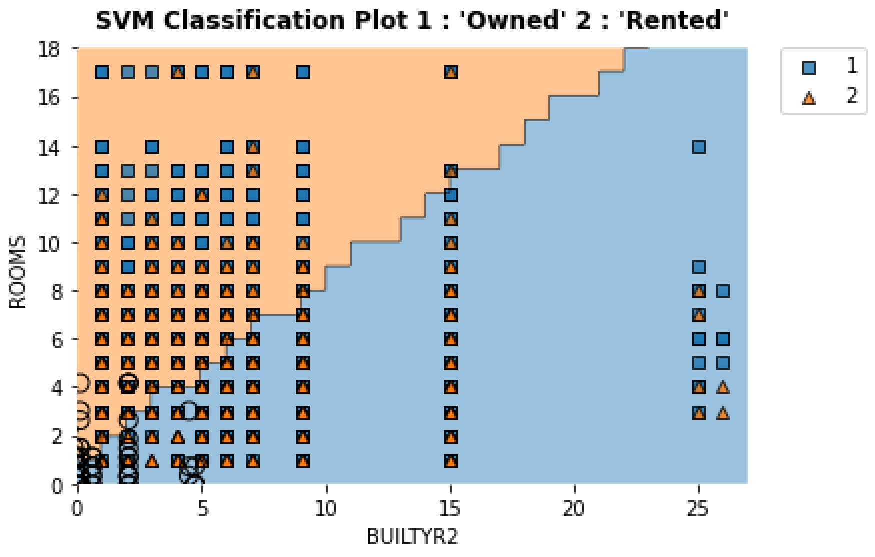
In [162…

```
def plot_lin(v1,v2):
    svmfit_1 = fit_two(v1,v2)
    # plot the support vector classifier
    plot_decision_regions(np.array(data_train[[v1,v2]]),np.array(data_train['OWNER
                          , legend = 2
                          , hide_spines =  True
                          , X_highlight=svmfit_1.support_vectors_ )
    plt.xlabel(v1)
    plt.ylabel(v2)
    plt.title('SVM Classification Plot 1 : \'Owned\' 2 : \'Rented\'', fontweight='
    plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
```

In [163…

```
plot_lin('BUILTYR2','ROOMS')
```

```
[LibSVM]Test Error: 0.49239999999999995
Test Accuracy: 0.5076
Training Error: 0.34119999999999995
Training Accuracy: 0.6588
```

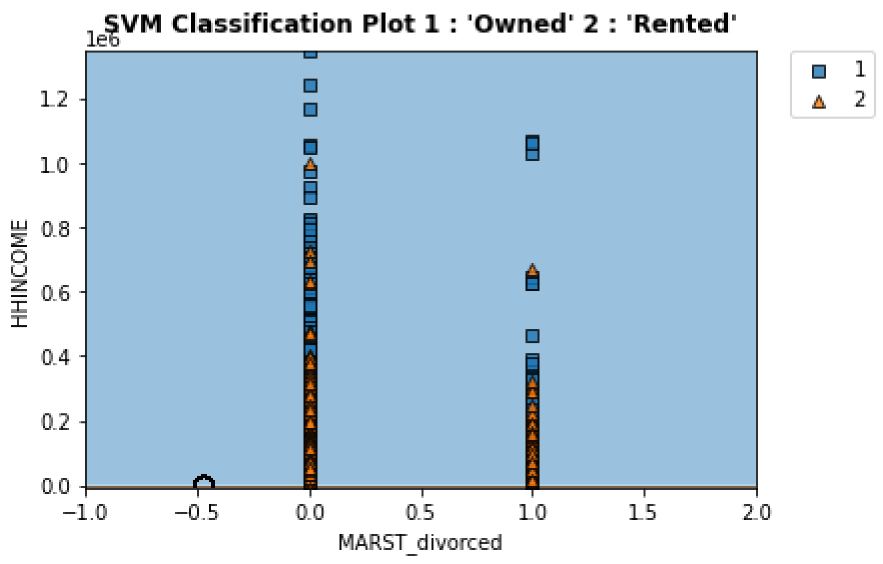


In [91]:

```
plot_lin('MARST_divorced','HHINCOME')
```

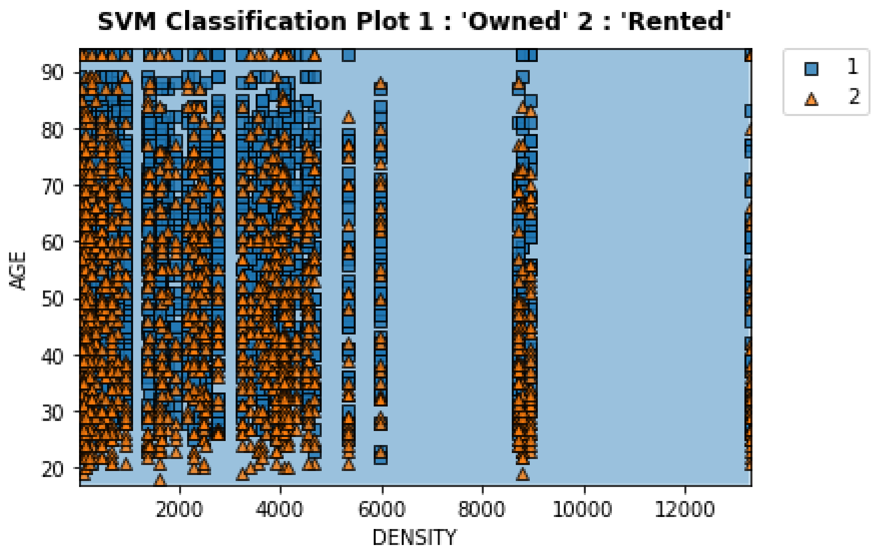
```
[LibSVM]Test Error: 0.33340000000000003
Test Accuracy: 0.6666
Training Error: 0.7070000000000001
Training Accuracy: 0.293
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



```
In [92]: plot_lin('DENSITY','AGE')
```

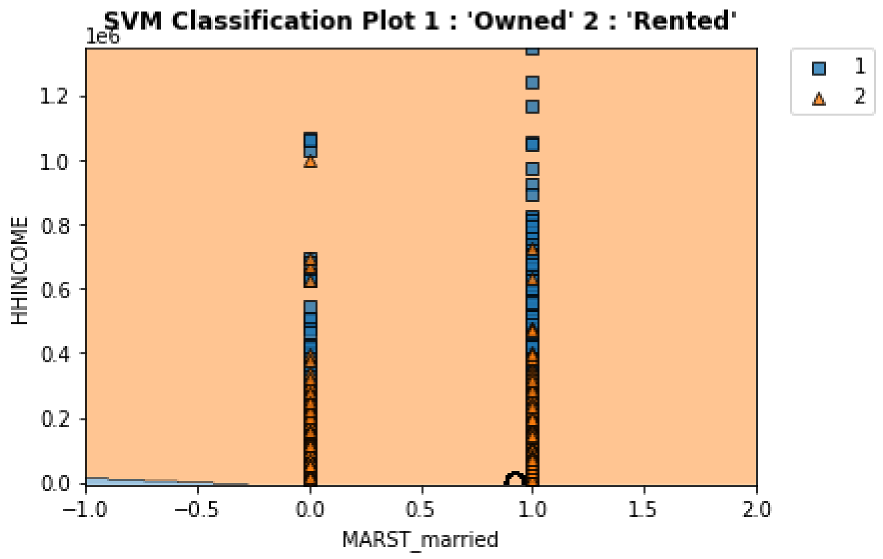
```
[LibSVM]Test Error: 0.6718
Test Accuracy: 0.3282
Training Error: 0.7078
Training Accuracy: 0.2922
```



```
In [93]: plot_lin('MARST_married','HHINCOME')
```

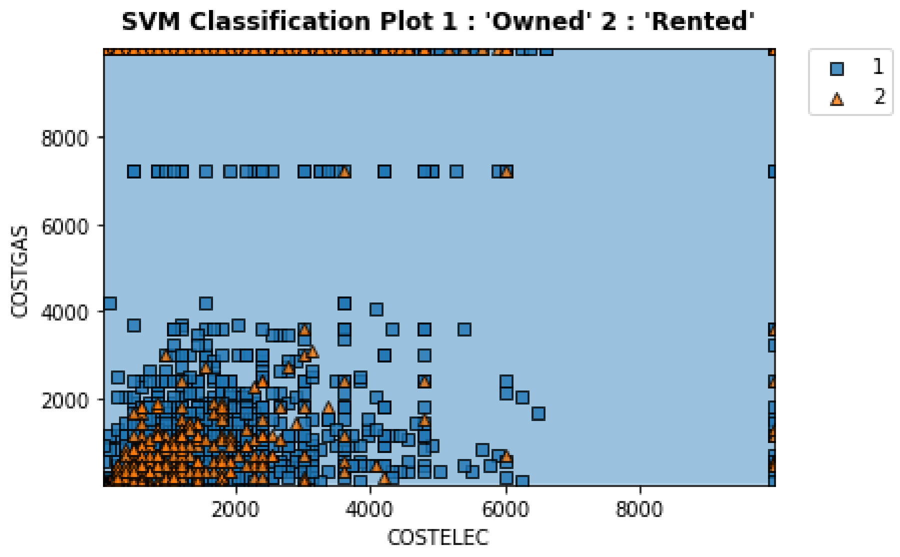
```
[LibSVM]Test Error: 0.3436
Test Accuracy: 0.6564
Training Error: 0.2922
Training Accuracy: 0.7078
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



```
In [94]: plot_lin('COSTELEC','COSTGAS')
```

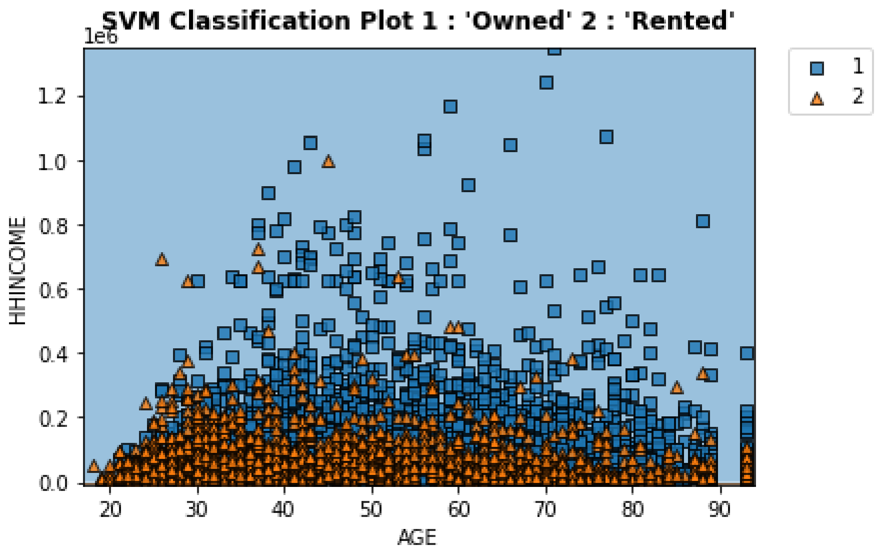
```
[LibSVM]Test Error: 0.2924
Test Accuracy: 0.7076
Training Error: 0.7078
Training Accuracy: 0.2922
```



```
In [95]: plot_lin('AGE','HHINCOME')
```

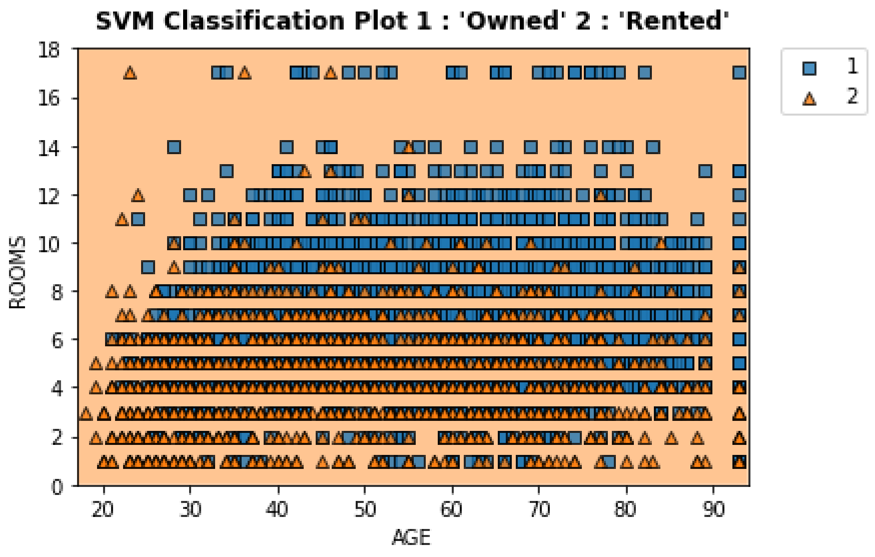
```
[LibSVM]Test Error: 0.7356
Test Accuracy: 0.2644
Training Error: 0.7078
Training Accuracy: 0.2922
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



In [121…
```
plot_lin('AGE','ROOMS')
```

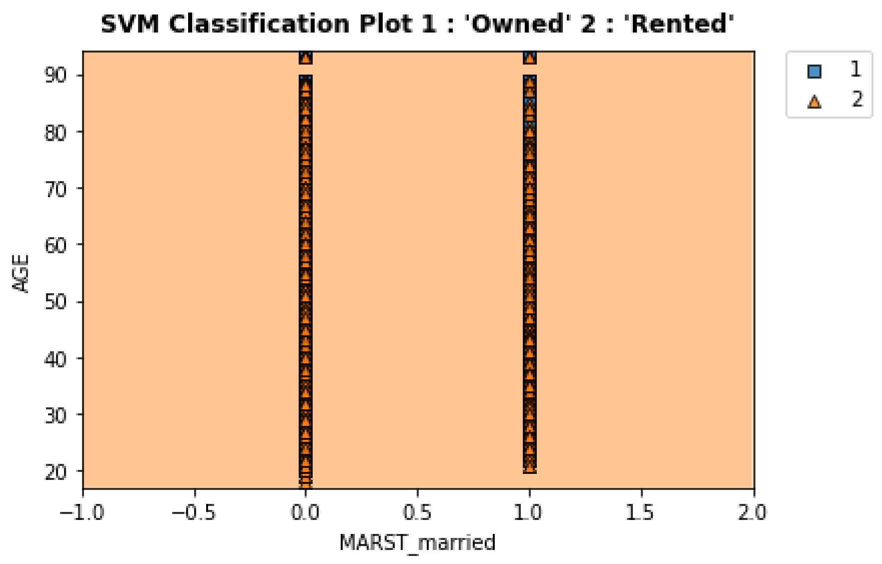
```
[LibSVM]Test Error: 0.6678
Test Accuracy: 0.3322
Training Error: 0.2922
Training Accuracy: 0.7078
```



In [96]:
```
plot_lin('MARST_married','AGE')
```

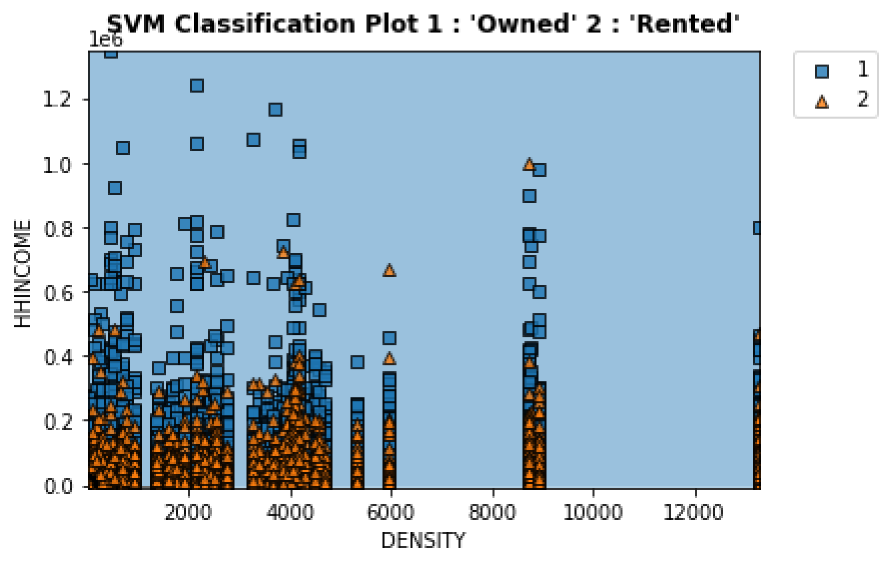
```
[LibSVM]Test Error: 0.2924
Test Accuracy: 0.7076
Training Error: 0.2922
Training Accuracy: 0.7078
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



In [97]:
```
plot_lin('DENSITY','HHINCOME')
```

```
[LibSVM]Test Error: 0.40659999999999996
Test Accuracy: 0.5934
Training Error: 0.7078
Training Accuracy: 0.2922
```



In my perspective, the a single linear hyper plane is not being able to bisect the region as we can see that lots of data points are clustered ine inside the other hence, it means that the SVM model being used is not able to capture the complex and non-linear relationship between COSTELEC and COSTGAS. This issue may arise when the SVM model is a linear classifier and is not flexible enough to capture the non-linear patterns in the data. To overcome this issue, one may consider using a non-linear SVM model or another type of model that can better handle non-linear relationships.

Radial Kernel

In [102…
```
def fit_two_rad(v1,v2):
    # Convert OWNERSHP to a factor variable
    data_train['OWNERSHP'] = data_train['OWNERSHP'].astype('category')
    data_test['OWNERSHP'] = data_test['OWNERSHP'].astype('category')
Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js
    scaler = StandardScaler()
    data_train_scaled = scaler.fit_transform(data_train[[v1,v2]])

    # Train SVM with linear kernel
    svmfit_3 = SVC(kernel='rbf', gamma=1, C=min_cost, cache_size=1000, verbose=Tru
    svmfit_3.fit(data_train_scaled, data_train['OWNERSHP'])

    # Make predictions on test set
    data_test_scaled = scaler.fit_transform(data_test[[v1,v2]])
    y_test_pred = svmfit_3.predict(data_test_scaled)

    #counfusion matrix
    confusion_matrix = pd.crosstab(index=y_test_pred, columns=data_test['OWNERSHP'
    print("\n\n",confusion_matrix)

    # Calculate training error and accuracy
    #test_error = 1 - accuracy_score(data_test['OWNERSHP'], y_test_pred)
    test_accuracy = accuracy_score(data_test['OWNERSHP'], y_test_pred)

    #print("Test Error:", test_error)
    print("Test Accuracy:", test_accuracy)
    # Make predictions on training set
    y_train_pred = svmfit_3.predict(data_train[[v1,v2]])

    # Calculate training error and accuracy
    #train_error = 1 - accuracy_score(data_train['OWNERSHP'], y_train_pred)
    train_accuracy = accuracy_score(data_train['OWNERSHP'], y_train_pred)

    #print("Training Error:", train_error)
    print("Training Accuracy:", train_accuracy)
    return svmfit_3
```

In [103…

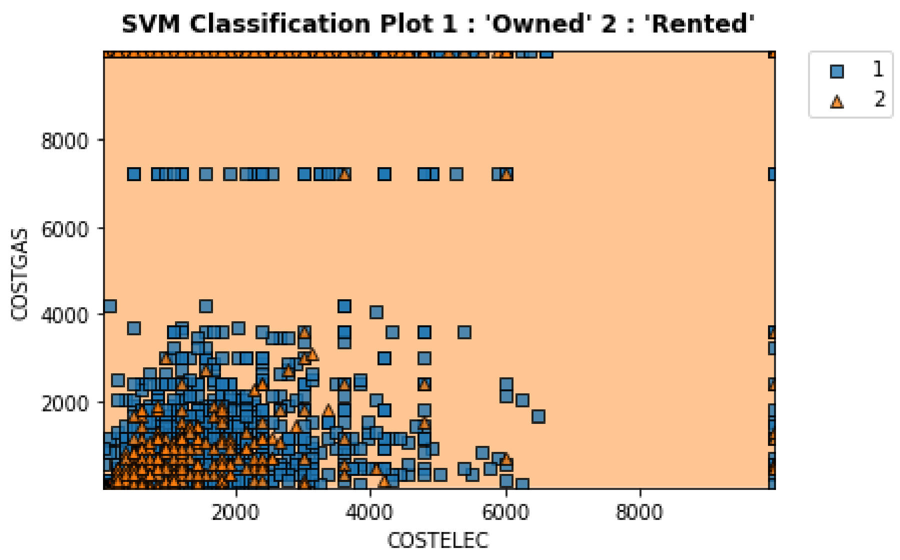
```
def plot_rad(v1,v2):
    svmfit_3 = fit_two_rad(v1,v2)
    # plot the support vector classifier
    plot_decision_regions(np.array(data_train[[v1,v2]]),np.array(data_train['OWNERS
                          , legend = 2
                          , hide_spines =  False
                          , X_highlight=svmfit_3.support_vectors_ )
    plt.xlabel(v1)
    plt.ylabel(v2)
    plt.title('SVM Classification Plot 1 : \'Owned\' 2 : \'Rented\'', fontweight=':
    plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
```

In [104…

```
plot_rad('COSTELEC','COSTGAS')
```

```
[LibSVM]

 OWNERSHP     1     2

1          1410   286
2          2128  1176
Test Accuracy: 0.5172
Training Accuracy: 0.7078
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

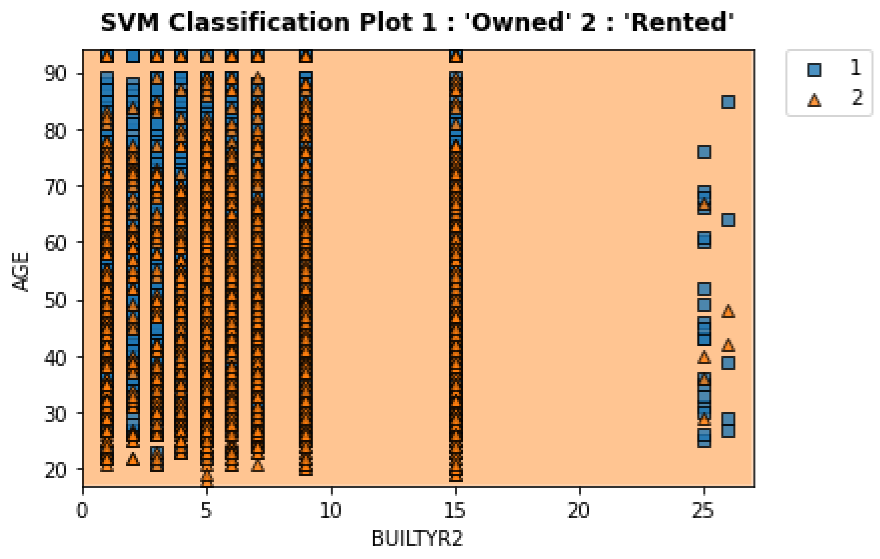


By the above plot we can see even with radial kernel we are not able to arrive at the distinct decision boundary hence we can say that these are not the strong predictors of Ownership classes.

In [187…

```
plot_rad('BUILTYR2','AGE')
```

```
[LibSVM]

 OWNERSHP     1     2

1          1722  1063
2          1816   399
Test Accuracy: 0.4242
Training Accuracy: 0.7078
```

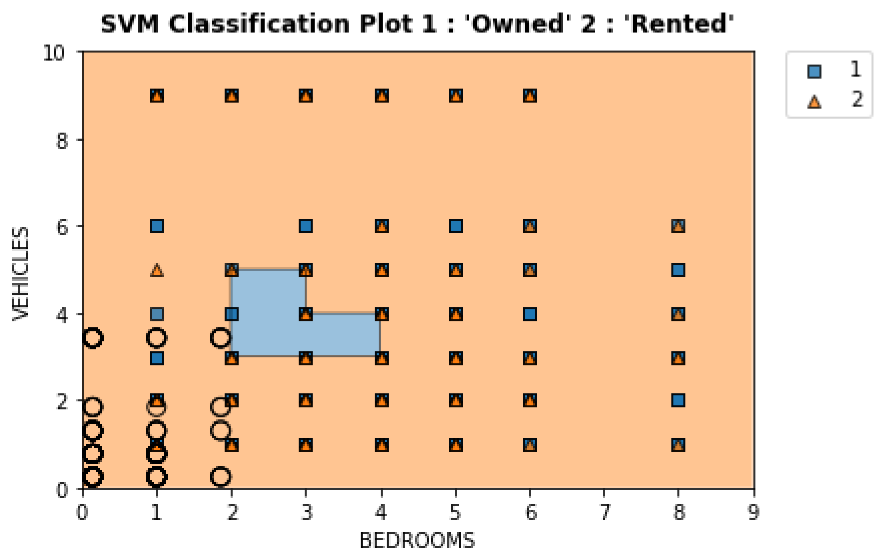


In [189…

```
plot_rad('BEDROOMS','VEHICLES')
```

```
[LibSVM]

 OWNERSHP     1     2

1          3510  1320
2            28   142
Test Accuracy: 0.7304
Training Accuracy: 0.6958
```

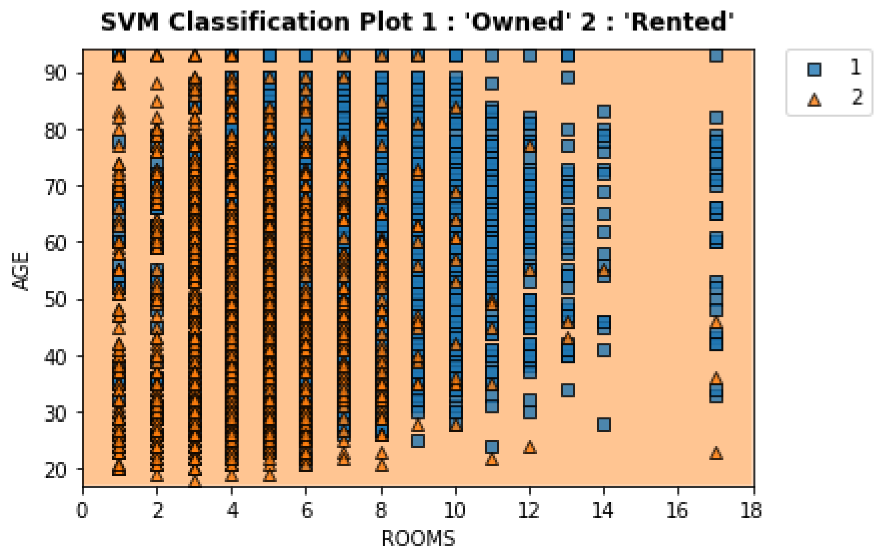
Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



In [122…

```
plot_rad('ROOMS','AGE')
```

```
[LibSVM]

 OWNERSHP      1     2

1          2379   824
2          1159   638
Test Accuracy: 0.6034
Training Accuracy: 0.7078
```

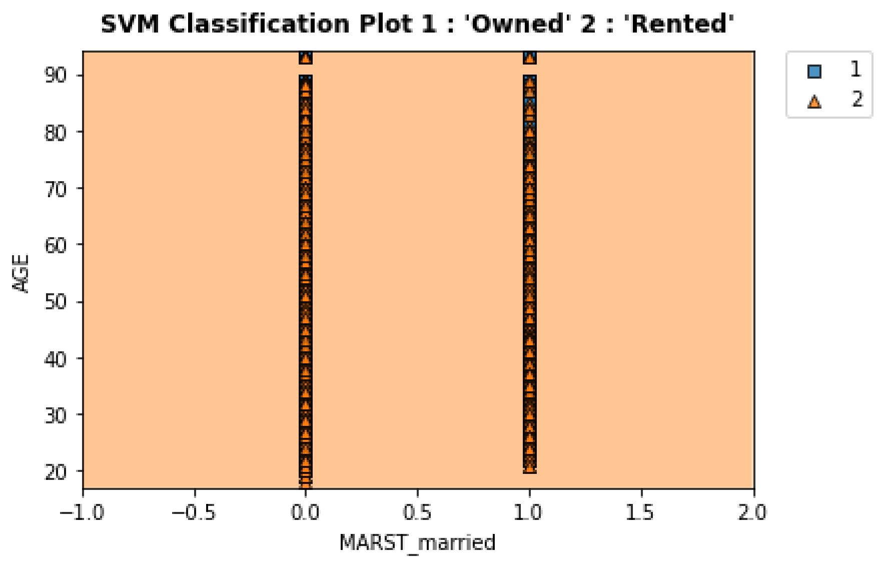


In [105…

```
plot_rad('MARST_married','AGE')
```

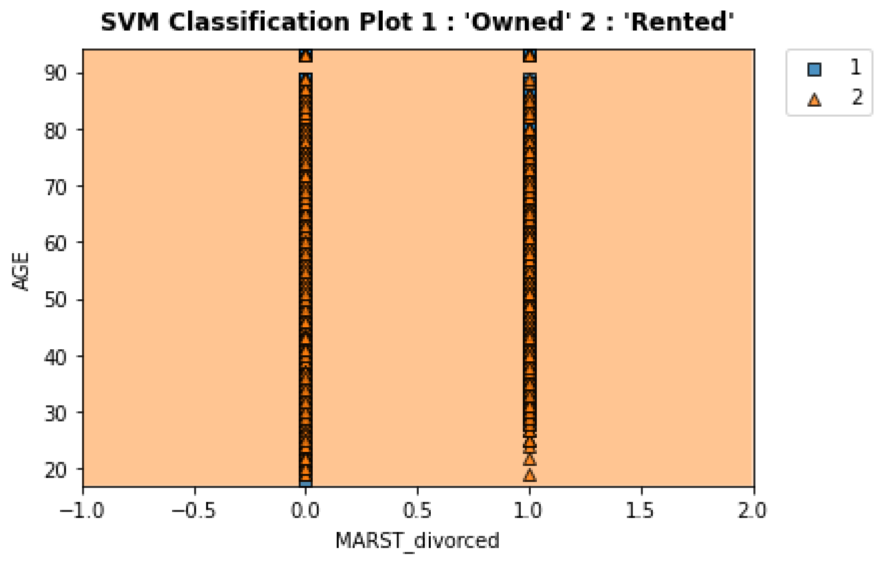
```
[LibSVM]

 OWNERSHP      1     2

1          2375   868
2          1163   594
Test Accuracy: 0.5938
Training Accuracy: 0.7078
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



```
In [55]: plot_rad('MARST_divorced','AGE')
```

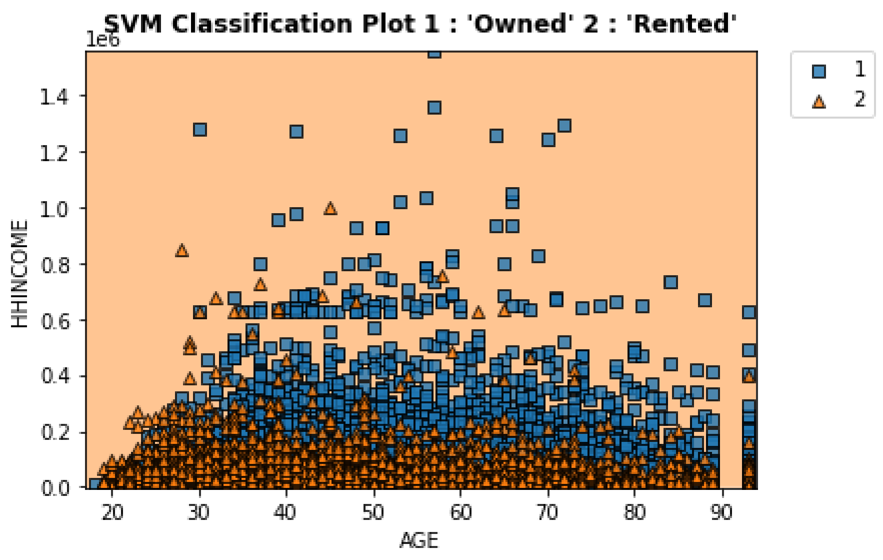
```
[LibSVM]

 OWNERSHP     1     2

1          5097  1691
2           208   504
Test Accuracy: 0.7468
Training Accuracy: 0.7117333333333333
```



```
In [56]: plot_rad('AGE','HHINCOME')
```

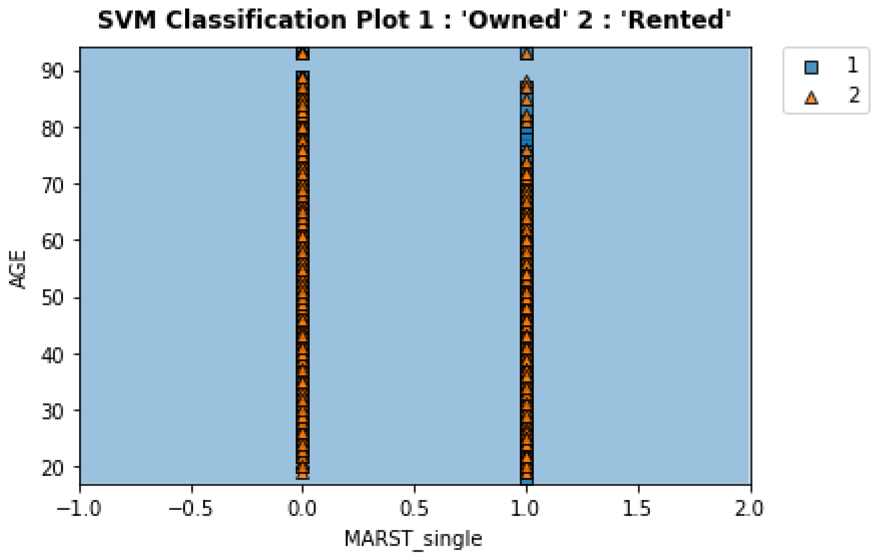
```
[LibSVM]

 OWNERSHP     1     2

1          4876  1308
2           429   887
Test Accuracy: 0.7684
Training Accuracy: 0.7117333333333333
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



In [57]:
```
plot_rad('MARST_single','AGE')
```

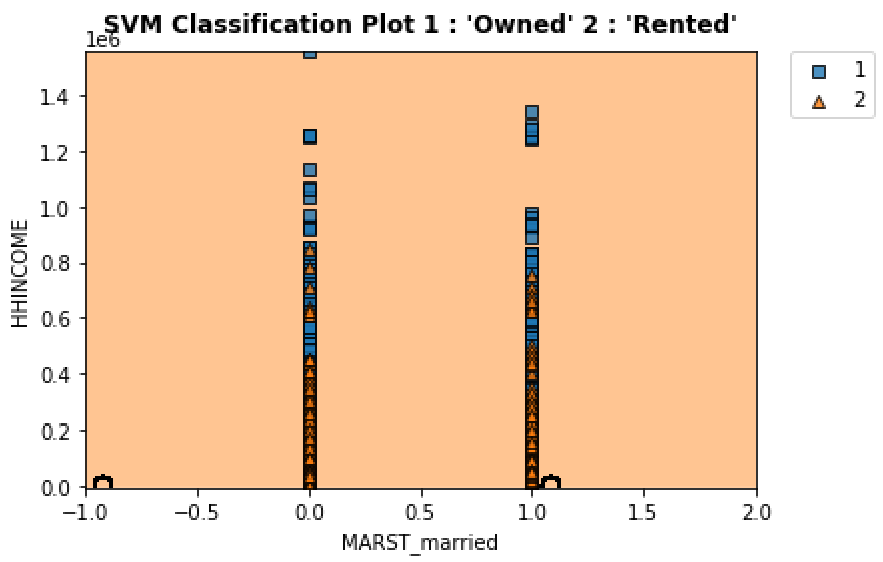
```
[LibSVM]

 OWNERSHP     1     2

1          4998  1472
2           307   723
Test Accuracy: 0.7628
Training Accuracy: 0.28826666666666667
```



In [18]:
```
plot_rad('MARST_married','HHINCOME')
```

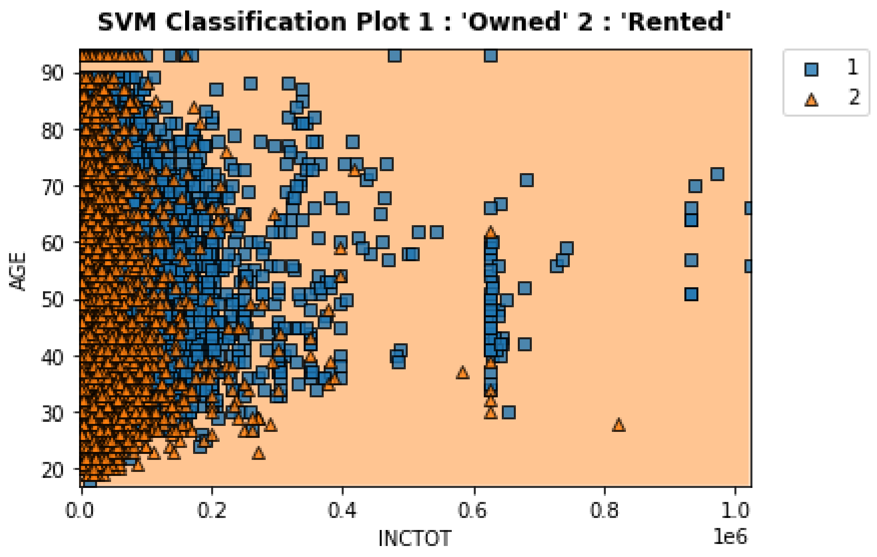
```
[LibSVM]

 OWNERSHP     1     2

1          6999  2067
2           384   550
Test Accuracy: 0.7549
Training Accuracy: 0.7348
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



In [59]:
```
plot_rad('INCTOT','AGE')
```

```
[LibSVM]

 OWNERSHP     1     2

1          5051  1586
2           254   609
Test Accuracy: 0.7546666666666667
Training Accuracy: 0.7117333333333333
```



Now, We perform cross-validation using `GridSearchCV()` to select the best choice of gamma and cost for an SVM with a radial kernel (Hyper parameter tuning):

In [ ]:
```
# svmfit = SVC(kernel='rbf', gamma=1, C=12, cache_size=1000, verbose=True, max_iter
# np.random.seed(2)

# grid_params = {'C': [0.1,10, 15, 50]
#                 ,'gamma': [0.5, 1, 2]}
# tune_out = GridSearchCV(svmfit, grid_params ,cv=10, scoring = 'accuracy')
# tune_out.fit(data_train[['COSTELEC','COSTGAS']],data_train[['OWNERSHP']])
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
[LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][L
ibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][Lib
SVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSV
M][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM]
[LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][L
ibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM]
```

Polynomial Kernel

In [184…

```
def fit_two_poly(v1,v2,dg):
    # Convert OWNERSHP to a factor variable
    data_train['OWNERSHP'] = data_train['OWNERSHP'].astype('category')
    data_test['OWNERSHP'] = data_test['OWNERSHP'].astype('category')

    # Scale the data
    scaler = StandardScaler()
    data_train_scaled = scaler.fit_transform(data_train[[v1,v2]])

    # Train SVM with polynomial kernel
    svmfit_4 = SVC(kernel='poly', gamma = 1, degree=dg, C=2, cache_size=1000, verb
    svmfit_4.fit(data_train_scaled, data_train['OWNERSHP'])

    # Make predictions on test set
    data_test_scaled = scaler.fit_transform(data_test[[v1,v2]])
    y_test_pred = svmfit_4.predict(data_test_scaled)

    #Test confusion matrix
    confusion_matrix = pd.crosstab(index=y_test_pred, columns=data_test['OWNERSHP'
    print("\n\n",confusion_matrix)

    # Calculate training error and accuracy
    #test_error = 1 - accuracy_score(data_test['OWNERSHP'], y_test_pred)
    test_accuracy = accuracy_score(data_test['OWNERSHP'], y_test_pred)

    #print("Test Error:", test_error)
    print("Test Accuracy:", test_accuracy)
    # Make predictions on training set
    y_train_pred = svmfit_4.predict(data_train[[v1,v2]])


    # Calculate training error and accuracy
    #train_error = 1 - accuracy_score(data_train['OWNERSHP'], y_train_pred)
    train_accuracy = accuracy_score(data_train['OWNERSHP'], y_train_pred)

    #print("Training Error:", train_error)
    print("Training Accuracy:", train_accuracy)
    return svmfit_4
```

In [185…

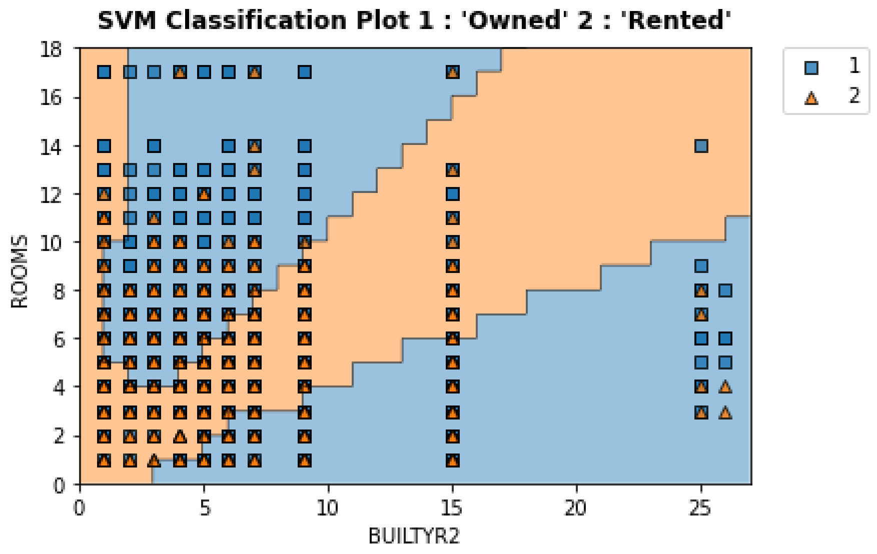
```
def plot_poly(v1,v2,dg):
    svmfit_4 = fit_two_poly(v1,v2,dg)
    # plot the support vector classifier
    plot_decision_regions(np.array(data_train[[v1,v2]]),np.array(data_train['OWNER
                          , legend = 2
                          , hide_spines =  False)
    plt.xlabel(v1)
    plt.ylabel(v2)
    plt.title('SVM Classification Plot 1 : \'Owned\' 2 : \'Rented\'', fontweight='
    plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
```

In [186…

```
plot_poly('BUILTYR2','ROOMS',3)
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

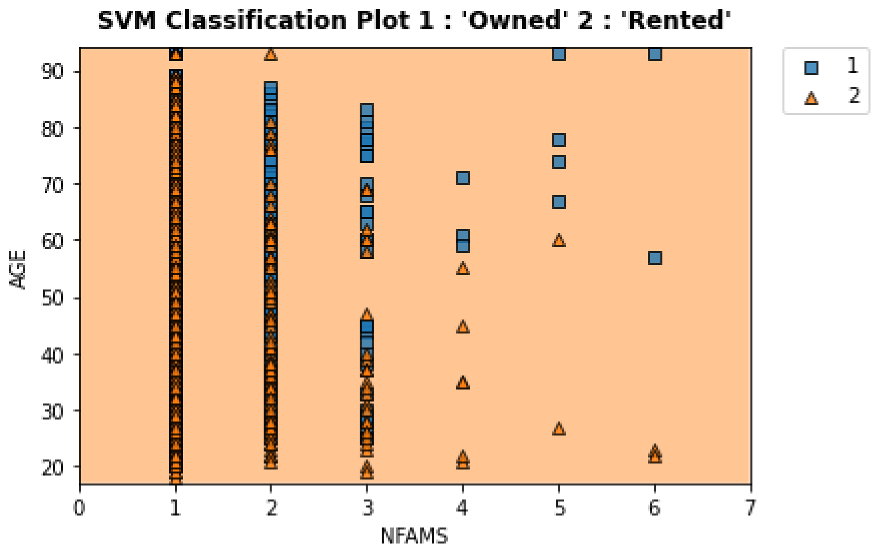
```
[LibSVM]

 OWNERSHP     1     2

1          3468  1292
2            70   170
Test Accuracy: 0.7276
Training Accuracy: 0.4352
```



In [191…

```
plot_rad('NFAMS','AGE')
```

```
[LibSVM]

 OWNERSHP     1     2

1          2280  628
2          1258  834
Test Accuracy: 0.6228
Training Accuracy: 0.7078
```

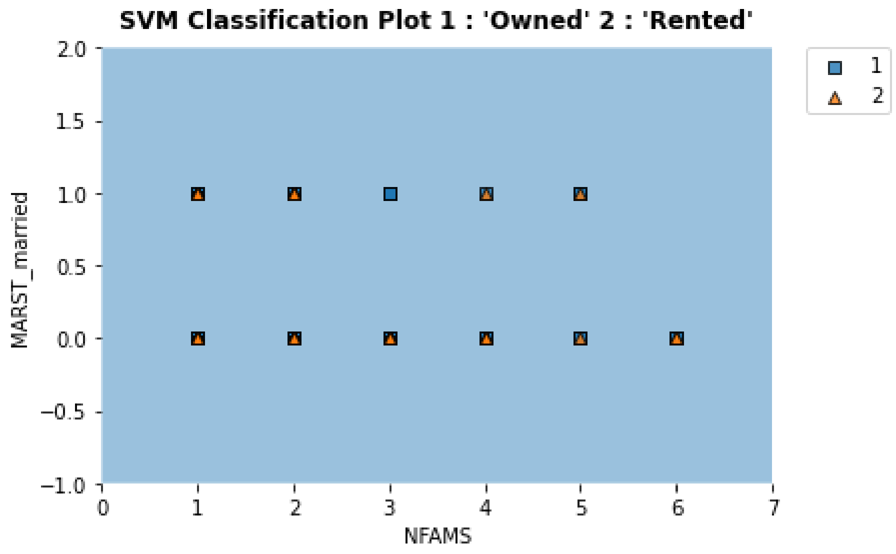


In [109…

```
plot_poly('NFAMS','MARST_married',3)
```

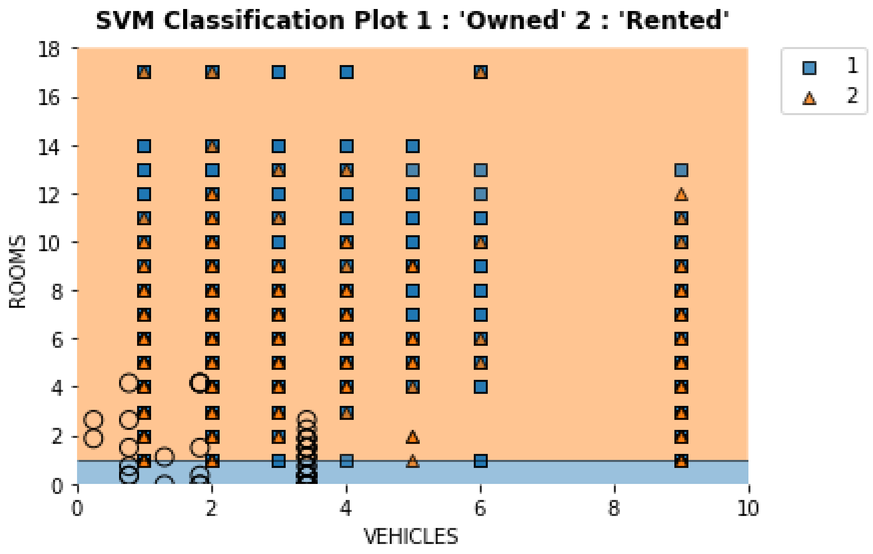
Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
[LibSVM]

 OWNERSHP     1     2

1            2     3
2         3536  1459
Test Accuracy: 0.2922
Training Accuracy: 0.2922
```



In [219…
```
plot_lin('VEHICLES','ROOMS')
```

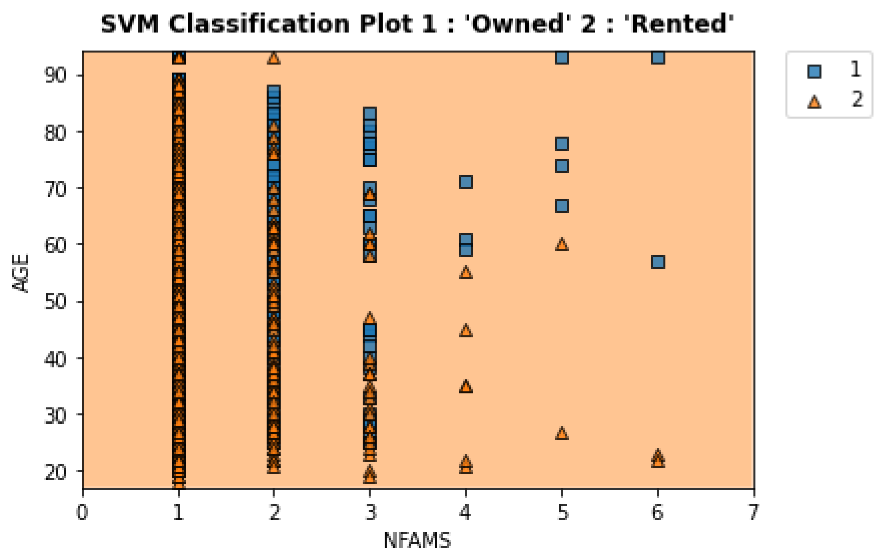
```
[LibSVM]Test Error: 0.4728
Test Accuracy: 0.5272
Training Error: 0.2922
Training Accuracy: 0.7078
```



In [198…
```
plot_rad('NFAMS','AGE')
```

```
[LibSVM]

 OWNERSHP     1    2

1         2280  628
2         1258  834
Test Accuracy: 0.6228
Training Accuracy: 0.7078
```

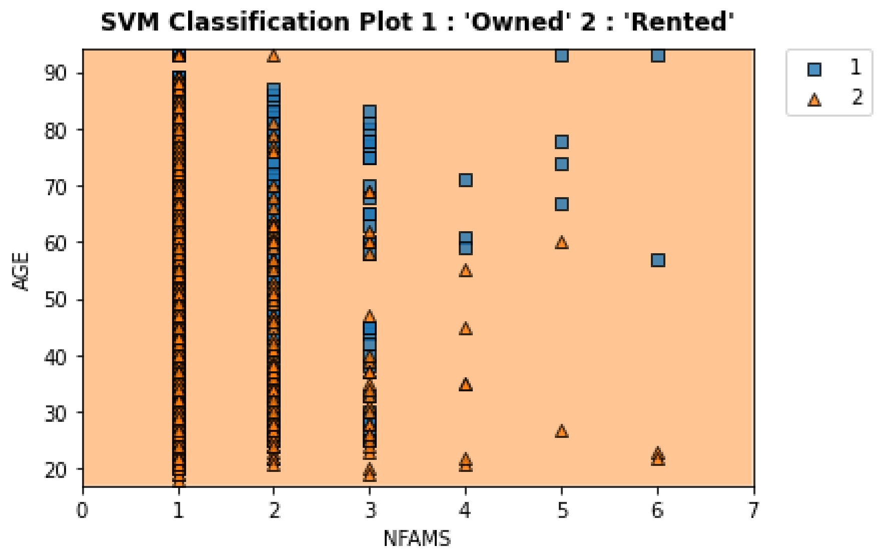
Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



In [199…

```
plot_poly('NFAMS','AGE',3)
```

```
[LibSVM]

 OWNERSHP     1     2

1            65    44
2          3473  1418
Test Accuracy: 0.2966
Training Accuracy: 0.7078
```

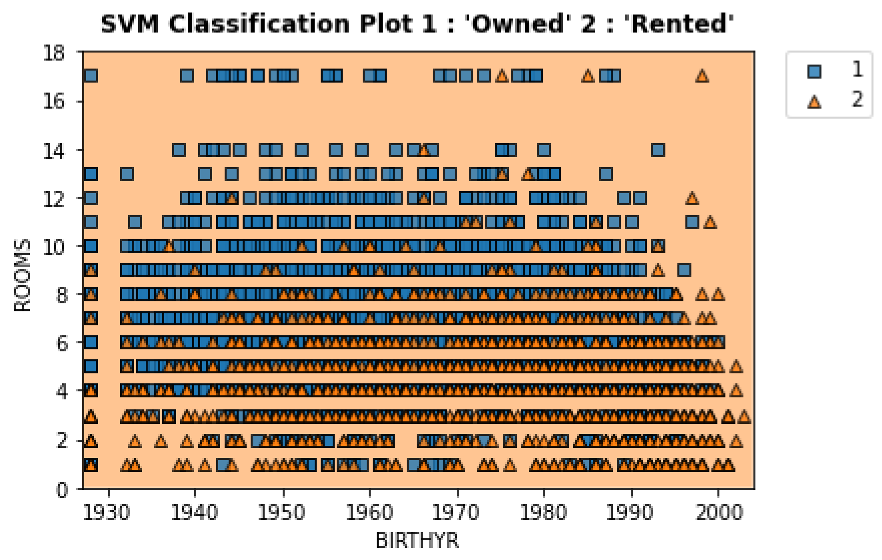


In [207…

```
plot_rad('BIRTHYR','ROOMS')
```

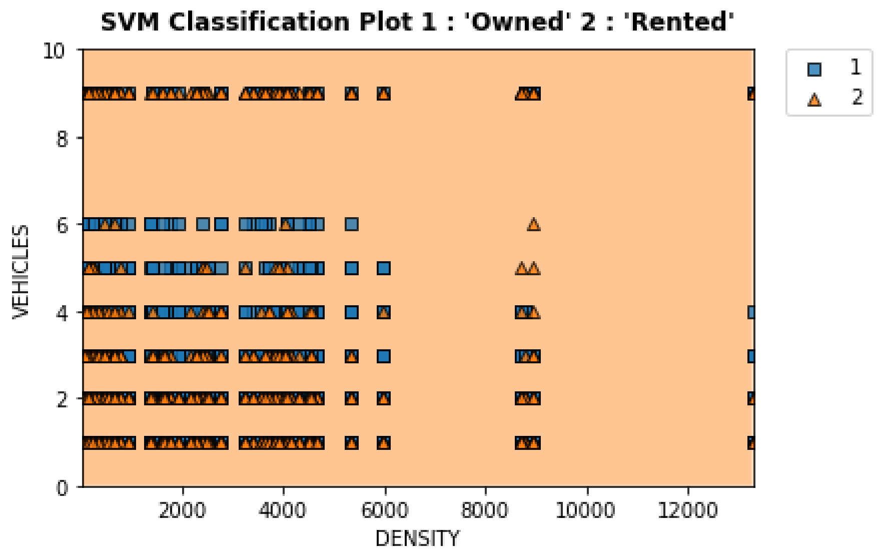
```
[LibSVM]

 OWNERSHP     1    2

1          2379  824
2          1159  638
Test Accuracy: 0.6034
Training Accuracy: 0.7078
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



```
In [217…  plot_poly('DENSITY','VEHICLES',3)
```

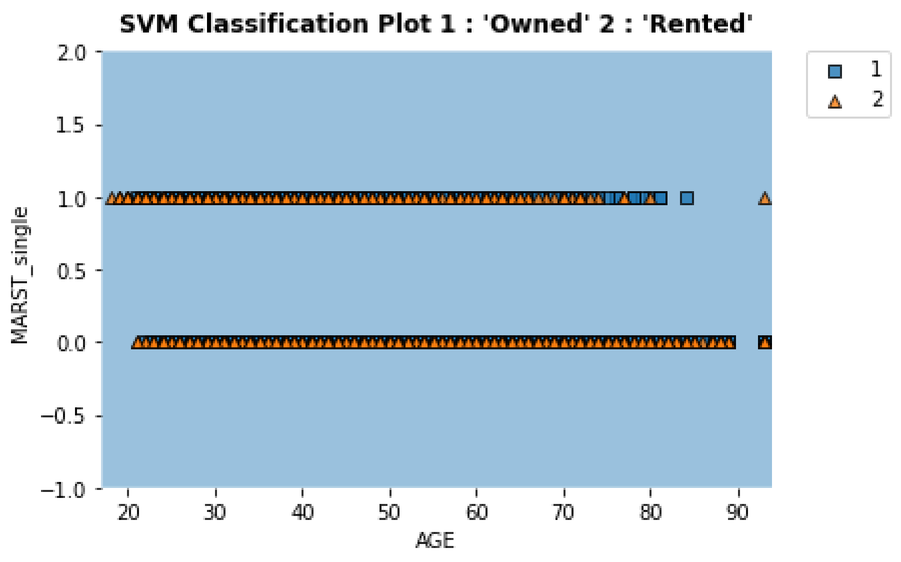
```
[LibSVM]

 OWNERSHP      1      2

1             25    115
2           3513   1347
Test Accuracy: 0.2744
Training Accuracy: 0.7078
```



```
In [111…  plot_poly('AGE','MARST_single',4)
```

```
[LibSVM]

 OWNERSHP      1      2

1           3306   1122
2            232    340
Test Accuracy: 0.7292
Training Accuracy: 0.2922
```

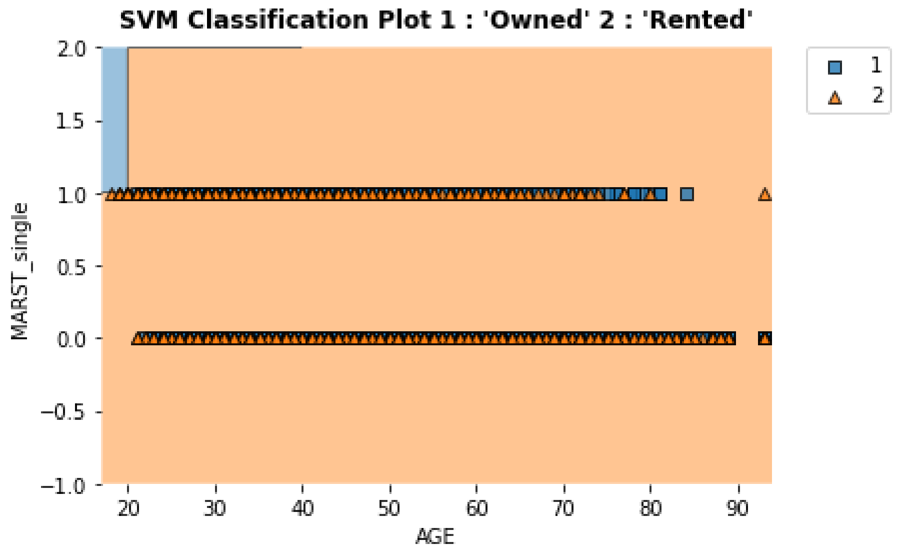
Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



In [112…

```
plot_poly('AGE','MARST_single',5)
```

```
[LibSVM]

 OWNERSHP      1     2

1          3349   966
2           189   496
Test Accuracy: 0.769
Training Accuracy: 0.7086
```

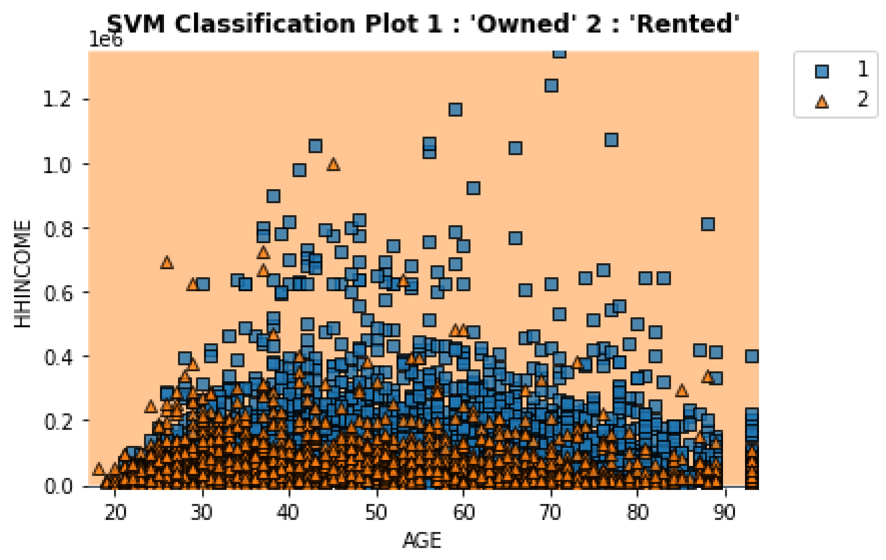


In [114…

```
plot_poly('AGE','HHINCOME',3)
```

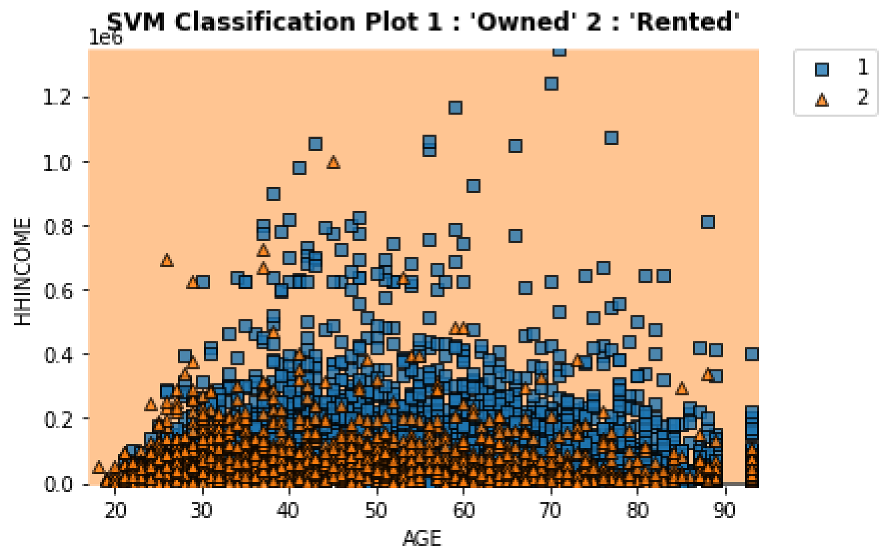
```
[LibSVM]

 OWNERSHP      1      2

1            63      8
2          3475   1454
Test Accuracy: 0.3034
Training Accuracy: 0.707
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



```
In [115…  plot_poly('AGE','HHINCOME',4)
```

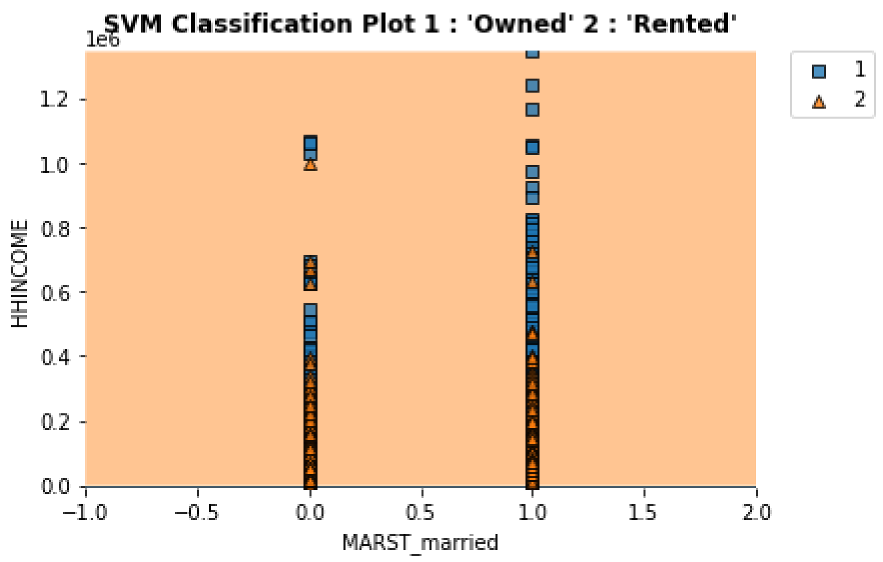
```
[LibSVM]

 OWNERSHP     1     2

1           155   224
2          3383  1238
Test Accuracy: 0.2786
Training Accuracy: 0.7078
```



```
In [116…  plot_poly('MARST_married','HHINCOME',3)
```

```
[LibSVM]

 OWNERSHP     1     2

1            60     8
2          3478  1454
Test Accuracy: 0.3028
Training Accuracy: 0.707
```

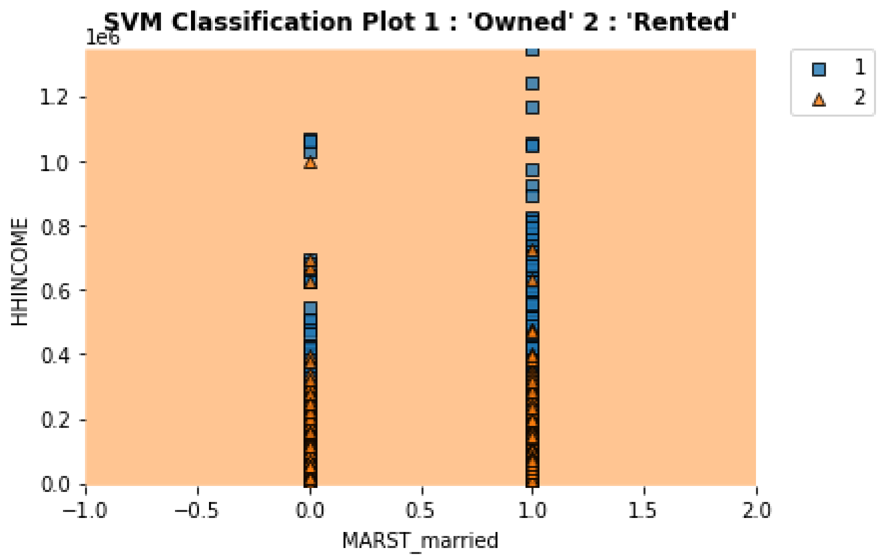
Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



In [117…

```
plot_poly('MARST_married','HHINCOME',4)
```

```
[LibSVM]

 OWNERSHP     1     2

1           12     0
2         3526  1462
Test Accuracy: 0.2948
Training Accuracy: 0.707
```

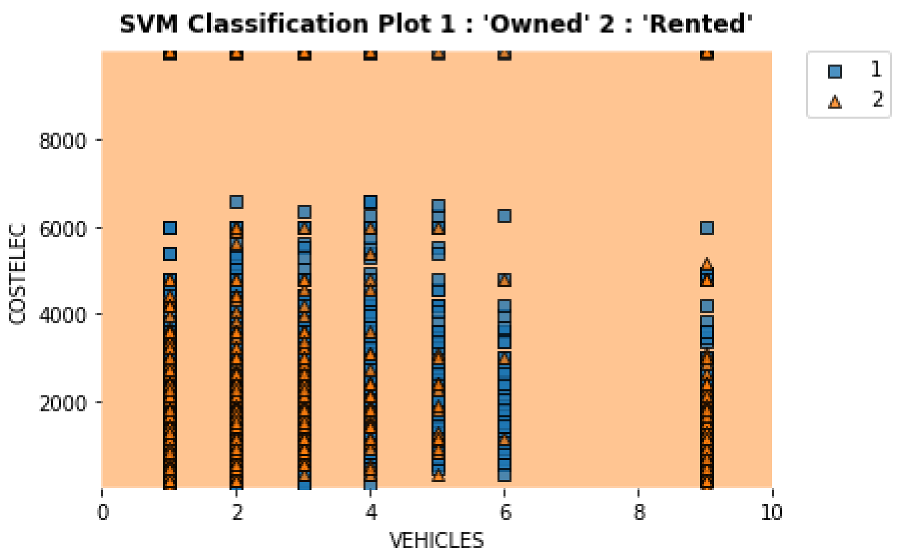


In [118…

```
plot_poly('VEHICLES','COSTELEC',4)
```

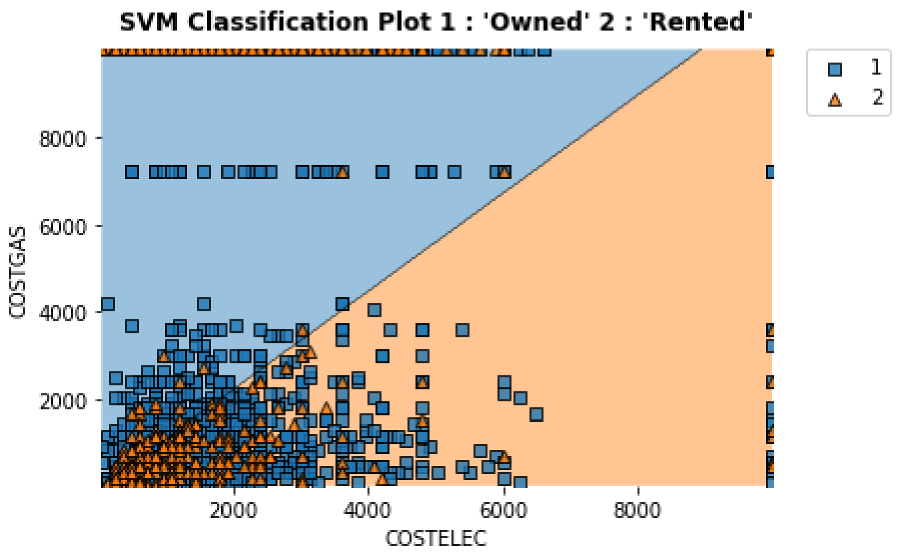
```
[LibSVM]

 OWNERSHP     1     2

1         3460  1255
2           78   207
Test Accuracy: 0.7334
Training Accuracy: 0.7078
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



```
In [120…  plot_poly('COSTELEC','COSTGAS',5)
```

```
[LibSVM]

 OWNERSHP      1      2

1             26      9
2           3512   1453
Test Accuracy: 0.2958
Training Accuracy: 0.462
```

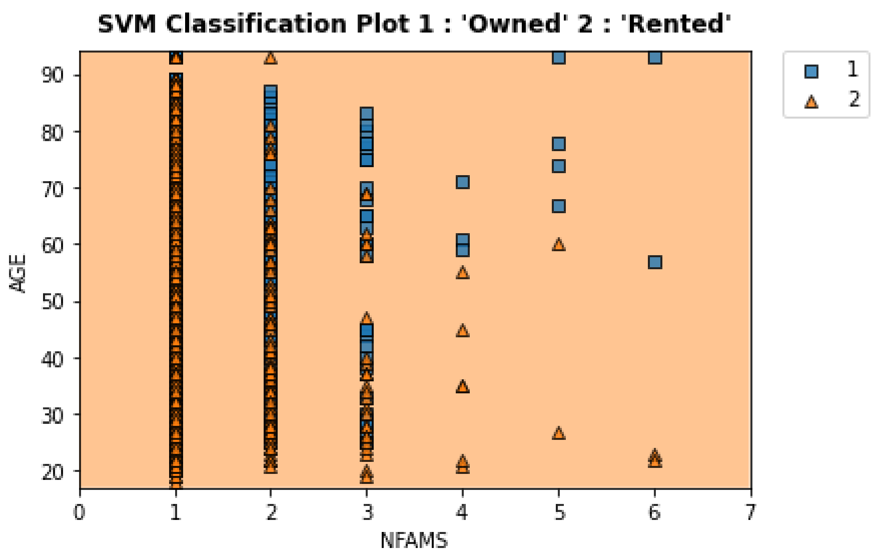


```
In [ ]:
```

```
In [223…  plot_rad('NFAMS','AGE')
```

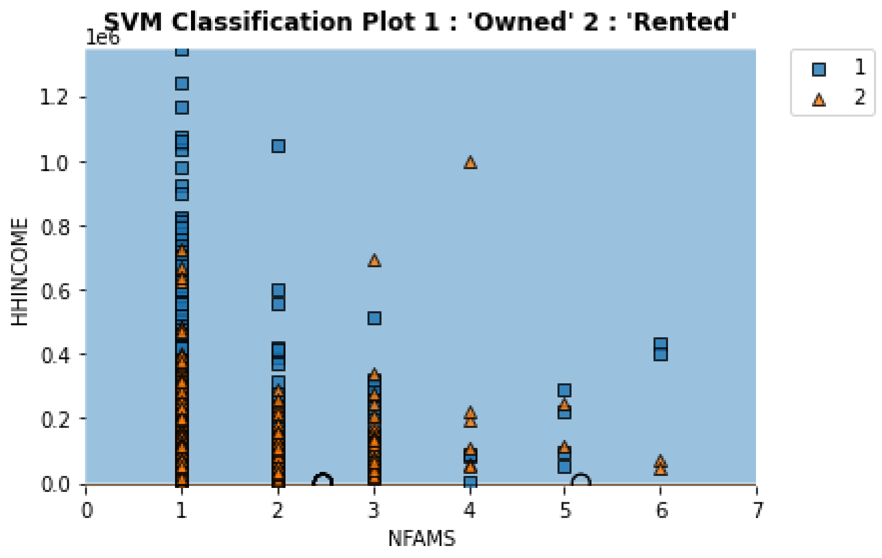
```
[LibSVM]

 OWNERSHP      1     2

1           2280   628
2           1258   834
Test Accuracy: 0.6228
Training Accuracy: 0.7078
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



In [229...
```
plot_lin('NFAMS','HHINCOME')
```

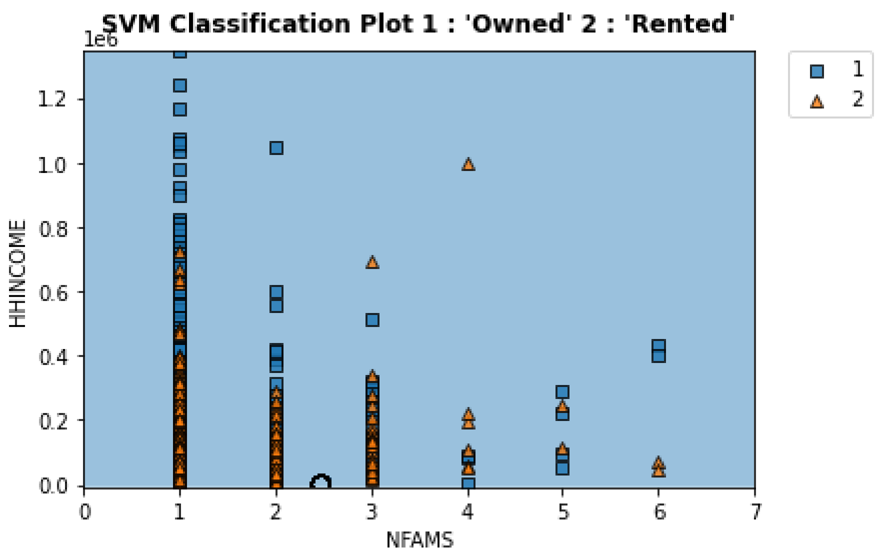
```
[LibSVM]Test Error: 0.2946
Test Accuracy: 0.7054
Training Error: 0.7070000000000001
Training Accuracy: 0.293
```



In [228...
```
plot_rad('NFAMS','HHINCOME')
```

```
[LibSVM]

 OWNERSHP     1     2

1          1001   148
2          2537  1314
Test Accuracy: 0.463
Training Accuracy: 0.2922
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



```
In [ ]:
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js