In [1]:
```python
# Importing the required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm
from ISLP.svm import plot as plot_svm
from sklearn.metrics import RocCurveDisplay
import sklearn.model_selection as skm
from ISLP import load_data, confusion_table
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import  StandardScaler
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.inspection import permutation_importance
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

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

In [2]:
```python
# Loading the dataset into a pandas DataFrame
Housing_data = pd.read_csv("Housing.csv")
```

In [3]:
```python
##Checking the dataset
Housing_data.head(5)
```

Out[3]:

	SERIAL	DENSITY	OWNERSHP	OWNERSHPD	COSTELEC	COSTGAS	COSTWATR	COST
0	1371772	920.0	1	13	9990	9993	360	
1	1371773	3640.9	2	22	1080	9993	1800	
2	1371773	3640.9	2	22	1080	9993	1800	
3	1371774	22.5	1	13	600	9993	9993	
4	1371775	3710.4	2	22	3600	9993	9997	

5 rows × 24 columns

In [4]:
```python
# Displaying the column names
Housing_data.columns
```

Out[4]:
```
Index(['SERIAL', 'DENSITY', 'OWNERSHP', 'OWNERSHPD', 'COSTELEC', 'COSTGAS',
       'COSTWATR', 'COSTFUEL', 'HHINCOME', 'VALUEH', 'ROOMS', 'BUILTYR2',
       'BEDROOMS', 'VEHICLES', 'NFAMS', 'NCOUPLES', 'PERNUM', 'PERWT', 'AGE',
       'MARST', 'BIRTHYR', 'EDUC', 'EDUCD', 'INCTOT'],
      dtype='object')
```

In [5]:
```python
## size of the data
Housing_data.shape
```

Out[5]: (75388, 24)

Data Pre-Processing

In [6]:
```
## Filtering the data based on the age > 16
Housing_data = Housing_data[Housing_data['AGE'] > 16]
```

In [7]:
```
## Checking the size after filter
Housing_data.shape
```

Out[7]: (60846, 24)

In [8]:
```
# subset the data based on married whose spouse is present
Married_subset = Housing_data[Housing_data['MARST'] == 1]
```

In [9]:
```
## checking the size of the subset data
Married_subset.shape
```

Out[9]: (33428, 24)

In [10]:
```
# Selecting columns needed for prediction
selected_data = Married_subset[['OWNERSHP', 'AGE', 'ROOMS','COSTELEC', 'COSTGAS','C
```

In [11]:
```
## checking the size
selected_data.shape
```

Out[11]: (33428, 10)

In [12]:
```
## checking the null values present
selected_data.isna().sum()
```

Out[12]:
```
OWNERSHP    0
AGE         0
ROOMS       0
COSTELEC    0
COSTGAS     0
COSTWATR    0
COSTFUEL    0
HHINCOME    0
BUILTYR2    0
VEHICLES    0
dtype: int64
```

LINEAR KERNEL

In [13]:
```
# Split data into X and y
X = selected_data.drop('OWNERSHP', axis=1)
y = selected_data['OWNERSHP']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta
```

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

In [14]:
```
# the range of C values
C_values = [0.01, 0.1, 1, 10, 50, 100]

for i in C_values:

    linear_svc = SVC(kernel='linear', C=i, verbose=True, max_iter=1000, random_stat
    linear_svc.fit(X_train_scaled, y_train)
    accuracy_linear = linear_svc.score(X_test_scaled, y_test)
    print(f"Linear SVM with C={i}: {accuracy_linear * 100:.2f}%")
```

```
[LibSVM]Linear SVM with C=0.01: 29.64%
[LibSVM]Linear SVM with C=0.1: 15.67%
[LibSVM]Linear SVM with C=1: 19.17%
[LibSVM]Linear SVM with C=10: 71.11%
[LibSVM]Linear SVM with C=50: 38.41%
[LibSVM]Linear SVM with C=100: 26.85%
```

In [15]:
```
from sklearn.feature_selection import SelectFromModel
from sklearn.ensemble import RandomForestClassifier
estimator = RandomForestClassifier(random_state=42)
# Perform feature selection
selector = SelectFromModel(estimator, threshold=-np.inf, max_features=9)
selector.fit(X_train_scaled, y_train)
# Transform the training and testing datasets
X_train_new = selector.transform(X_train_scaled)
X_test_new = selector.transform(X_test_scaled)
```

In [16]:
```
C_values = [0.01, 0.1, 1, 10, 50, 100]

for i in C_values:
    linear_svc_ = SVC(kernel='linear', C=i, cache_size=1000, verbose = True, max_it
    linear_svc_.fit(X_train_new, y_train)

    accuracy = linear_svc_.score(X_test_new, y_test)
    print(f"Linear SVM with C={i}: {accuracy * 100:.2f}%")
```

```
[LibSVM]Linear SVM with C=0.01: 84.62%
[LibSVM]Linear SVM with C=0.1: 84.62%
[LibSVM]Linear SVM with C=1: 86.40%
[LibSVM]Linear SVM with C=10: 39.66%
[LibSVM]Linear SVM with C=50: 59.50%
[LibSVM]Linear SVM with C=100: 30.95%
```

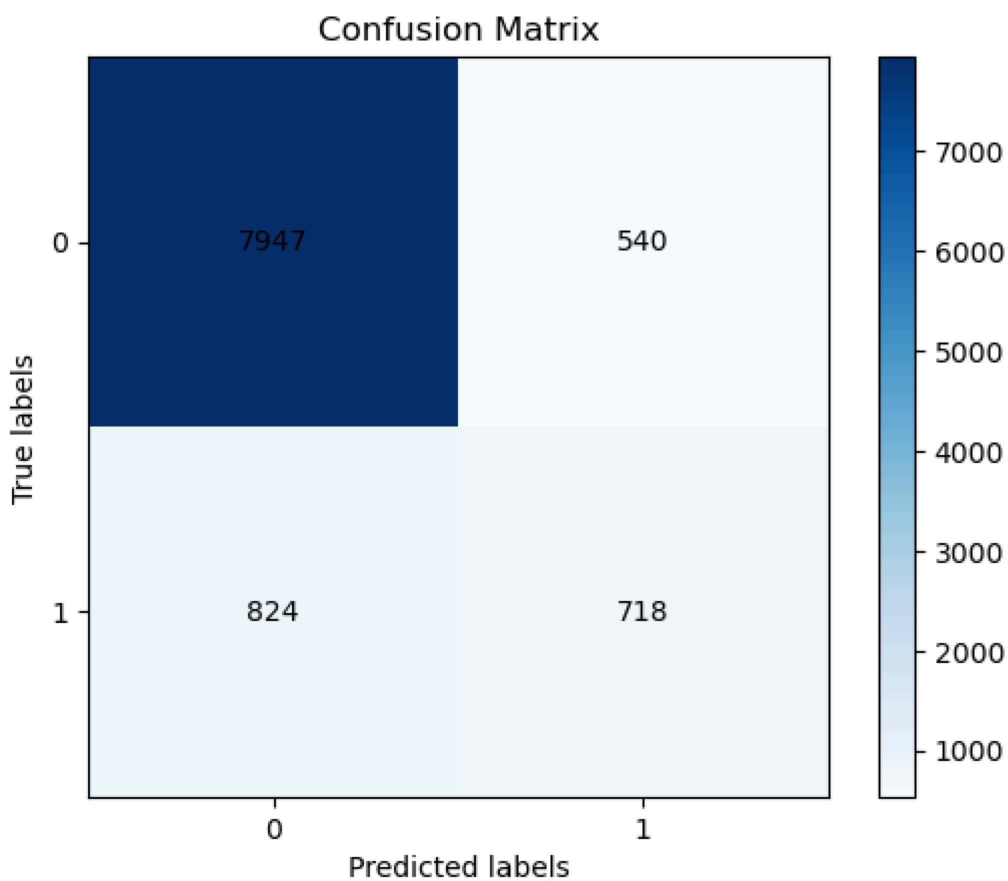
In [17]:
```
## fitting svc linear kernel model at c=1
```

In [18]:
```
linear_svc_f = SVC(kernel='linear', C=1, cache_size=1000, verbose = True, max_iter
linear_svc_f.fit(X_train_new, y_train)
accuracy = linear_svc_f.score(X_test_new, y_test)
print("Linear SVM with C=1: {:.2f}%".format(accuracy * 100))
```

```
[LibSVM]Linear SVM with C=1: 86.40%
```

In [19]:

```python
y_pred = linear_svc_f.predict(X_test_new)

# Calculating the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Plotting the confusion matrix using imshow
plt.imshow(conf_matrix, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.colorbar()
tick_marks = np.arange(len(conf_matrix))
plt.xticks(tick_marks, tick_marks)
plt.yticks(tick_marks, tick_marks)
plt.xlabel('Predicted labels')
plt.ylabel('True labels')

# Displaying the values in the cells
for i in range(len(conf_matrix)):
    for j in range(len(conf_matrix)):
        plt.text(j, i, conf_matrix[i, j], horizontalalignment='center', verticalali

plt.show()
```



In [20]:

```python
from sklearn.inspection import permutation_importance
result = permutation_importance(linear_svc_f, X_test_new, y_test, n_repeats=10, ran
importances = result.importances_mean
```

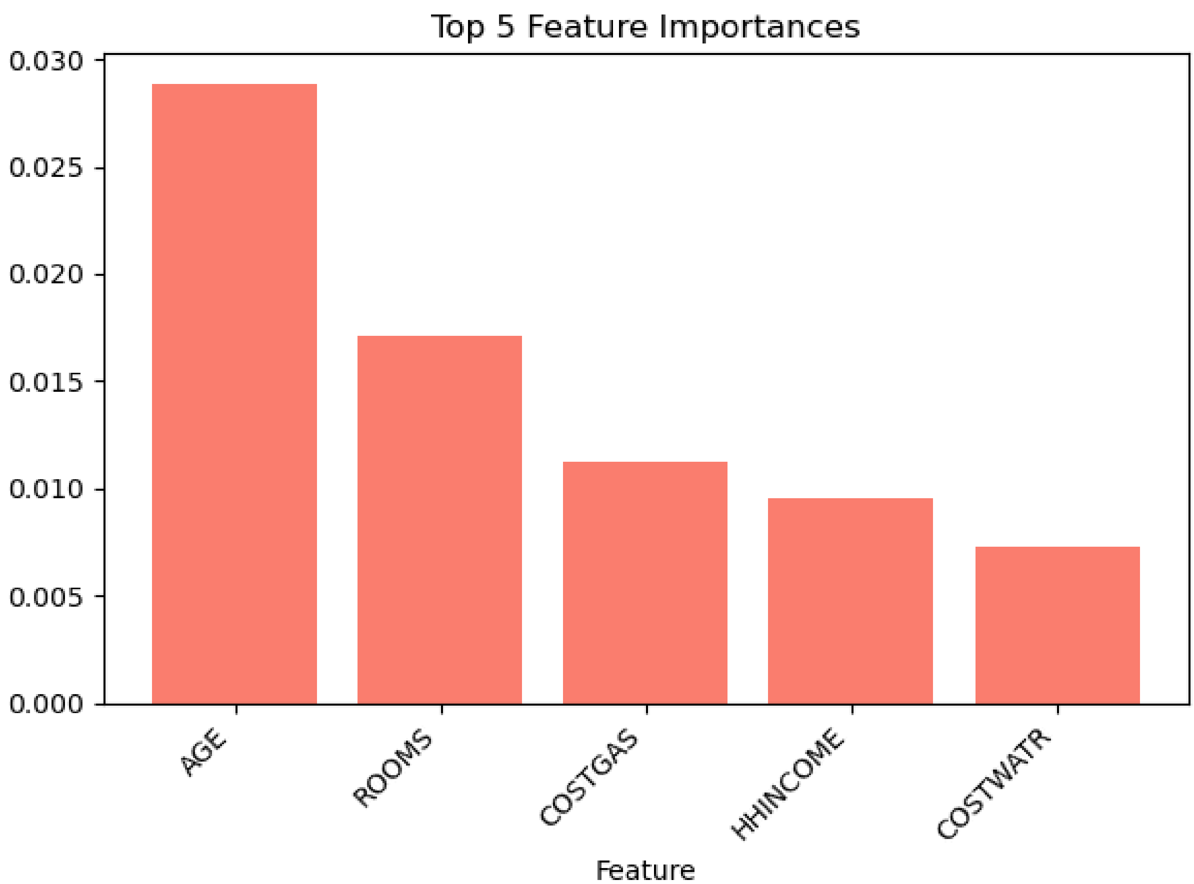
In [21]:
```
importances
```

Out[21]:
```
array([0.02880646, 0.01712035, 0.00053844, 0.01123741, 0.00728886,
       0.00464653, 0.00958221, 0.00054841, 0.00287167])
```

In [22]:
```
# Map importances to feature names
importances_dict = {feature: importance for feature, importance in zip(X.columns, i

# Sort features by importance in ascending order
sorted_importances = sorted(importances_dict.items(), key=lambda x: x[1], reverse=T

# Display features with their importance in ascending order
print("Features with their importance (in descending order):")
for feature, importance in sorted_importances:
    print(f"{feature}: {importance}")
```

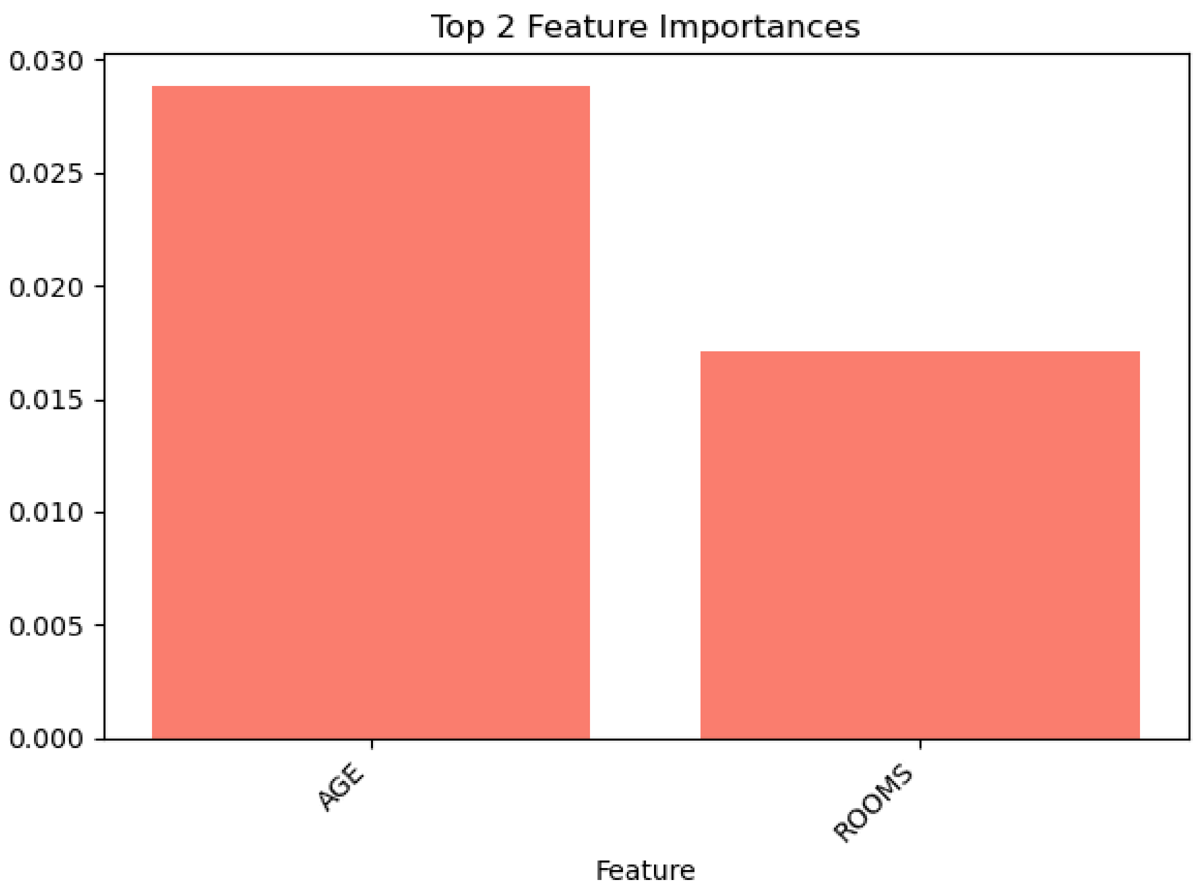
```
Features with their importance (in descending order):
AGE: 0.028806461262339
ROOMS: 0.017120350982151776
COSTGAS: 0.011237411506630757
HHINCOME: 0.009582211586399436
COSTWATR: 0.007288862299331922
COSTFUEL: 0.004646525077275887
VEHICLES: 0.002871672150762794
BUILTYR2: 0.0005484096121248338
COSTELEC: 0.0005384385282680149
```

In [23]:
```
import matplotlib.pyplot as plt

# Extract features and importances from sorted_importances
features = [item[0] for item in sorted_importances[:5]]
importances = [item[1] for item in sorted_importances[:5]]

# Plot the bar chart with a different color
plt.bar(features, importances, color='salmon')

# Set title and labels
plt.title('Top 5 Feature Importances')
plt.xlabel('Feature')
plt.ylabel('Importance')

# Rotate x-axis labels for better readability
plt.xticks(rotation=45, ha='right')

# Remove y-axis label and grid lines
plt.ylabel('')
plt.grid(False)

# Show plot
plt.tight_layout()
plt.show()
```



In [24]:

```
import matplotlib.pyplot as plt

# Extract features and importances for top 2 predictors
top_features = [item[0] for item in sorted_importances[:2]]
top_importances = [item[1] for item in sorted_importances[:2]]

# Plot the bar chart with a different color
plt.bar(top_features, top_importances, color='salmon')

# Set title and labels
plt.title('Top 2 Feature Importances')
plt.xlabel('Feature')
plt.ylabel('Importance')

# Rotate x-axis labels for better readability
plt.xticks(rotation=45, ha='right')

# Remove y-axis label and grid lines
plt.ylabel('')
plt.grid(False)

# Show plot
plt.tight_layout()
plt.show()
```

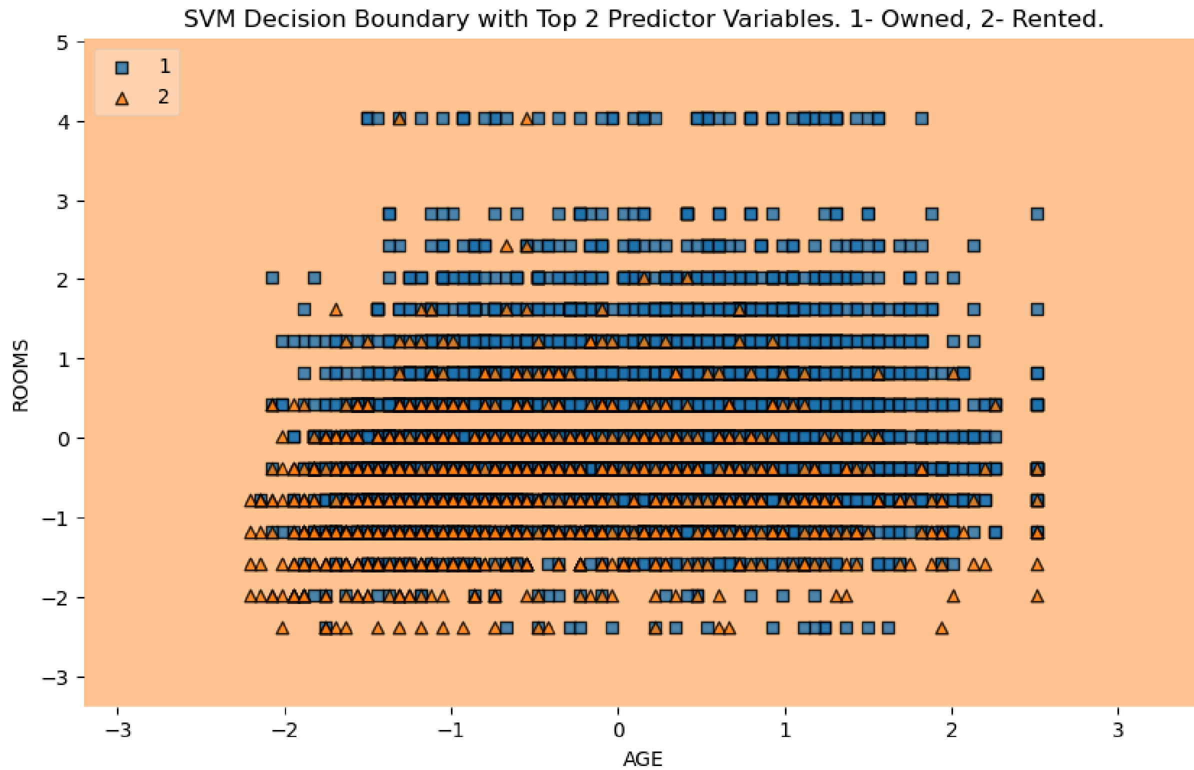


In [25]:
```
top_features
```

Out[25]: ['AGE', 'ROOMS']

In [26]:
```
from mlxtend.plotting import plot_decision_regions
# Filter the dataset to keep only the top 2 predictor variables
X_new = X[top_features]

# Split the data into training and testing sets
X_train_2, X_test_2, y_train_2, y_test_2 = train_test_split(X_new, y, test_size=0.2
# Scale the training and testing data
scaler = StandardScaler()
X_train_2_n = scaler.fit_transform(X_train_2)
X_test_2_n = scaler.transform(X_test_2)
# Train the best model with the top 2 predictor variables
linear_svc_2 = SVC(kernel='linear', C= 1, cache_size=1000, verbose=True, max_iter=1
linear_svc_2.fit(X_train_2_n, y_train_2)

# Plot the decision boundary
plt.figure(figsize=(10, 6))
plot_decision_regions(X_test_2_n, y_test_2.to_numpy(), clf=linear_svc_2, legend=2)
plt.xlabel(top_features[0])
plt.ylabel(top_features[1])
plt.title('SVM Decision Boundary with Top 2 Predictor Variables. 1- Owned, 2- Rente
plt.show()
```

[LibSVM]



Polynomial

In [27]:

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

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta

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

In [28]:

```
C_values = [0.01, 0.1, 1, 10, 100, 1000]
degrees = [2, 3, 4]  # Degrees to iterate over

for i in C_values:
    for degree in degrees:  # Loop over different degrees
        polynomial_svc = SVC(kernel='poly', degree=degree, C=i, cache_size=1000, ve
        polynomial_svc.fit(X_train_scaled, y_train)
        accuracy = polynomial_svc.score(X_test_scaled, y_test)

        print(f"Polynomial SVM with C={i} and degree = {degree}: {accuracy * 100:.2
```

```
[LibSVM]Polynomial SVM with C=0.01 and degree = 2: 85.72%
[LibSVM]Polynomial SVM with C=0.01 and degree = 3: 87.25%
[LibSVM]Polynomial SVM with C=0.01 and degree = 4: 87.00%
[LibSVM]Polynomial SVM with C=0.1 and degree = 2: 85.74%
[LibSVM]Polynomial SVM with C=0.1 and degree = 3: 88.21%
[LibSVM]Polynomial SVM with C=0.1 and degree = 4: 87.71%
[LibSVM]Polynomial SVM with C=1 and degree = 2: 85.67%
[LibSVM]Polynomial SVM with C=1 and degree = 3: 88.49%
[LibSVM]Polynomial SVM with C=1 and degree = 4: 88.56%
[LibSVM]Polynomial SVM with C=10 and degree = 2: 82.23%
[LibSVM]Polynomial SVM with C=10 and degree = 3: 22.02%
[LibSVM]Polynomial SVM with C=10 and degree = 4: 26.17%
[LibSVM]Polynomial SVM with C=100 and degree = 2: 32.57%
[LibSVM]Polynomial SVM with C=100 and degree = 3: 22.01%
[LibSVM]Polynomial SVM with C=100 and degree = 4: 24.97%
[LibSVM]Polynomial SVM with C=1000 and degree = 2: 28.38%
[LibSVM]Polynomial SVM with C=1000 and degree = 3: 24.29%
[LibSVM]Polynomial SVM with C=1000 and degree = 4: 27.71%
```

In [29]:

```python
estimator = RandomForestClassifier(random_state=42)
# Perform feature selection
selector = SelectFromModel(estimator, threshold=-np.inf, max_features=9)
selector.fit(X_train_scaled, y_train)
X_train_new = selector.transform(X_train_scaled) # Transform the training and testi
X_test_new = selector.transform(X_test_scaled)
```

In [30]:

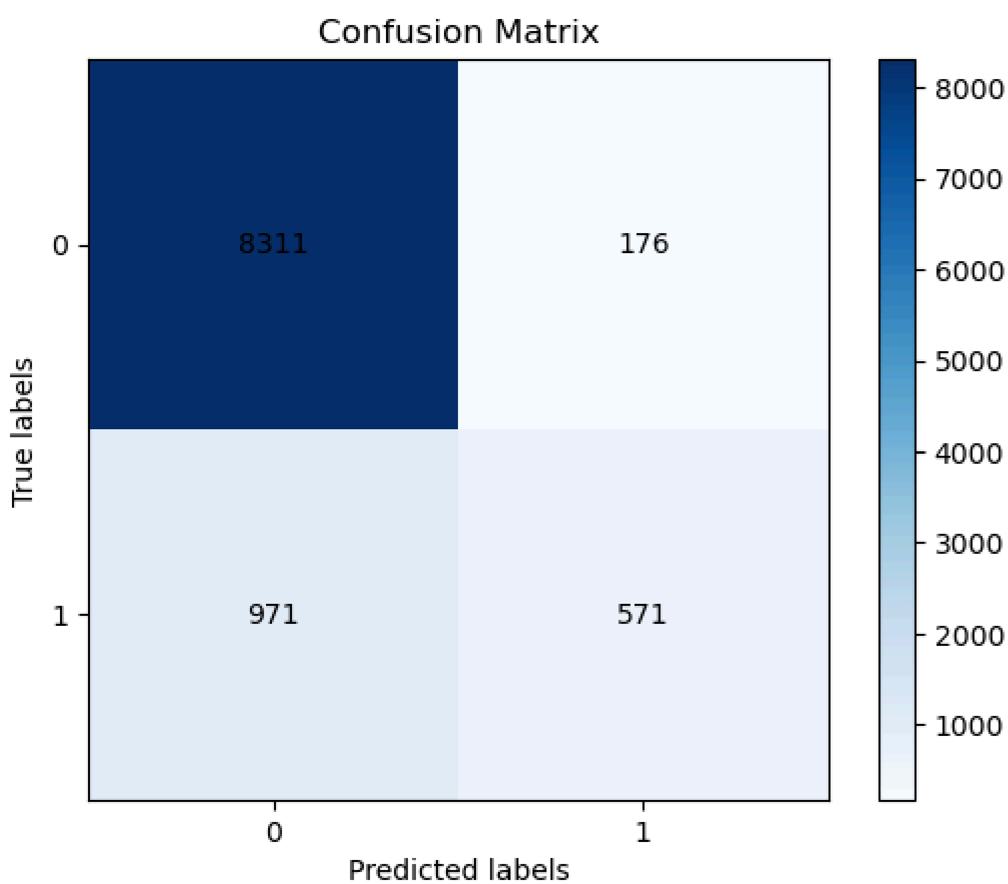
```python
C_values = [0.01, 0.1, 1, 10, 100, 1000]
degrees = [2, 3, 4]  # Degrees to iterate over

for i in C_values:
    for degree in degrees:  # Loop over different degrees
        polynomial_svc = SVC(kernel='poly', degree=degree, C=i, cache_size=1000, ve
        polynomial_svc.fit(X_train_new, y_train)

        # Evaluate the model on the testing data and print the accuracy score
        accuracy = polynomial_svc.score(X_test_new, y_test)
        print(f"Polynomial SVM with C={i} and degree = {degree}: {accuracy * 100:.2
```

```
[LibSVM]Polynomial SVM with C=0.01 and degree = 2: 85.72%
[LibSVM]Polynomial SVM with C=0.01 and degree = 3: 87.25%
[LibSVM]Polynomial SVM with C=0.01 and degree = 4: 87.00%
[LibSVM]Polynomial SVM with C=0.1 and degree = 2: 85.74%
[LibSVM]Polynomial SVM with C=0.1 and degree = 3: 88.21%
[LibSVM]Polynomial SVM with C=0.1 and degree = 4: 87.71%
[LibSVM]Polynomial SVM with C=1 and degree = 2: 85.67%
[LibSVM]Polynomial SVM with C=1 and degree = 3: 88.49%
[LibSVM]Polynomial SVM with C=1 and degree = 4: 88.56%
[LibSVM]Polynomial SVM with C=10 and degree = 2: 82.23%
[LibSVM]Polynomial SVM with C=10 and degree = 3: 22.02%
[LibSVM]Polynomial SVM with C=10 and degree = 4: 26.17%
[LibSVM]Polynomial SVM with C=100 and degree = 2: 32.57%
[LibSVM]Polynomial SVM with C=100 and degree = 3: 22.01%
[LibSVM]Polynomial SVM with C=100 and degree = 4: 24.97%
[LibSVM]Polynomial SVM with C=1000 and degree = 2: 28.38%
[LibSVM]Polynomial SVM with C=1000 and degree = 3: 24.29%
[LibSVM]Polynomial SVM with C=1000 and degree = 4: 27.71%
```

In [31]:
```
polynomial_svc_f = SVC(kernel='poly', degree=4, C=1, cache_size=1000, verbose=True,
polynomial_svc_f.fit(X_train_new, y_train)
accuracy = polynomial_svc_f.score(X_test_new, y_test)
print("Polynomial SVM with C=1 and degree=4: {:.2f}%".format(accuracy * 100))
```

```
[LibSVM]Polynomial SVM with C=1 and degree=4: 88.56%
```

In [32]:
```
y_pred = polynomial_svc_f.predict(X_test_new)

# Calculating the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Plotting the confusion matrix using imshow
plt.imshow(conf_matrix, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.colorbar()
tick_marks = np.arange(len(conf_matrix))
plt.xticks(tick_marks, tick_marks)
plt.yticks(tick_marks, tick_marks)
plt.xlabel('Predicted labels')
plt.ylabel('True labels')

# Displaying the values in the cells
for i in range(len(conf_matrix)):
    for j in range(len(conf_matrix)):
        plt.text(j, i, conf_matrix[i, j], horizontalalignment='center', verticalali

plt.show()
```

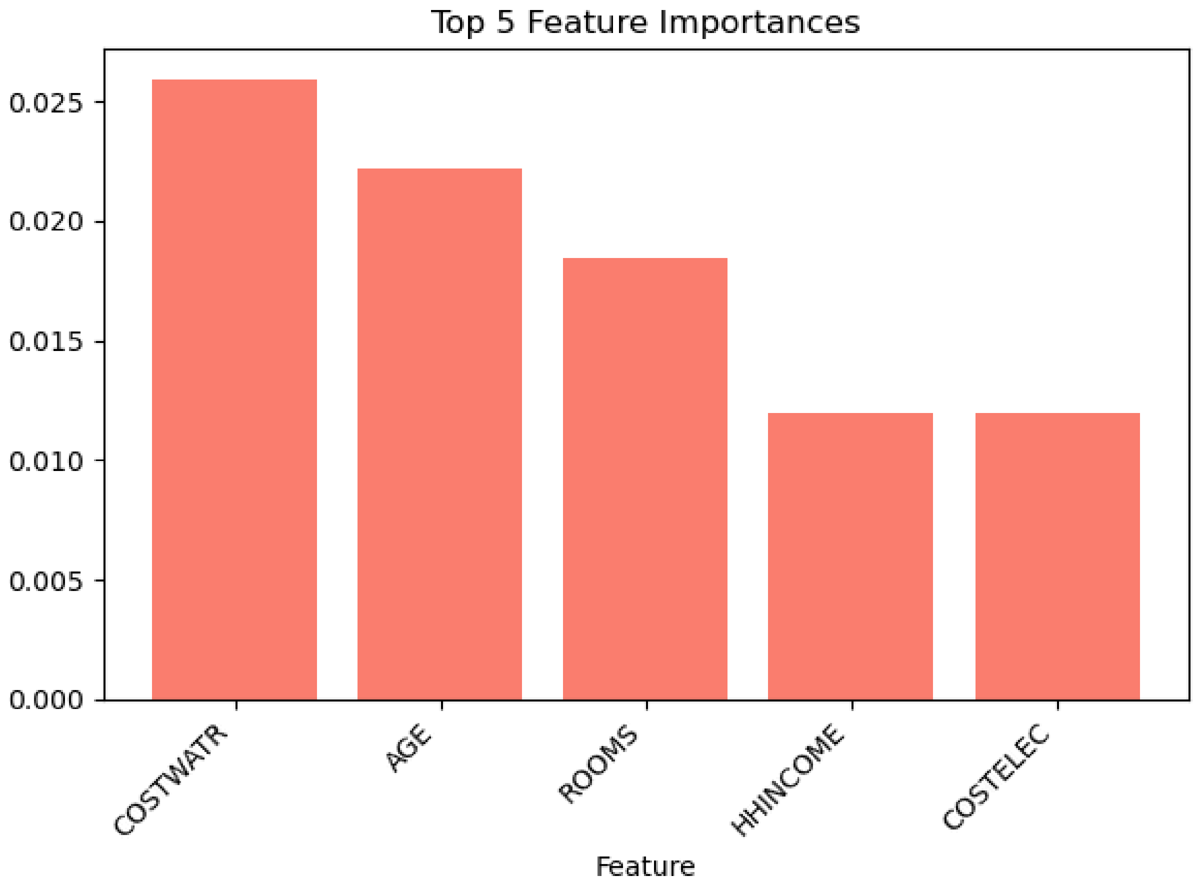


In [33]:
```
from sklearn.inspection import permutation_importance
result = permutation_importance(polynomial_svc_f, X_test_new, y_test, n_repeats=10,
importances = result.importances_mean
```

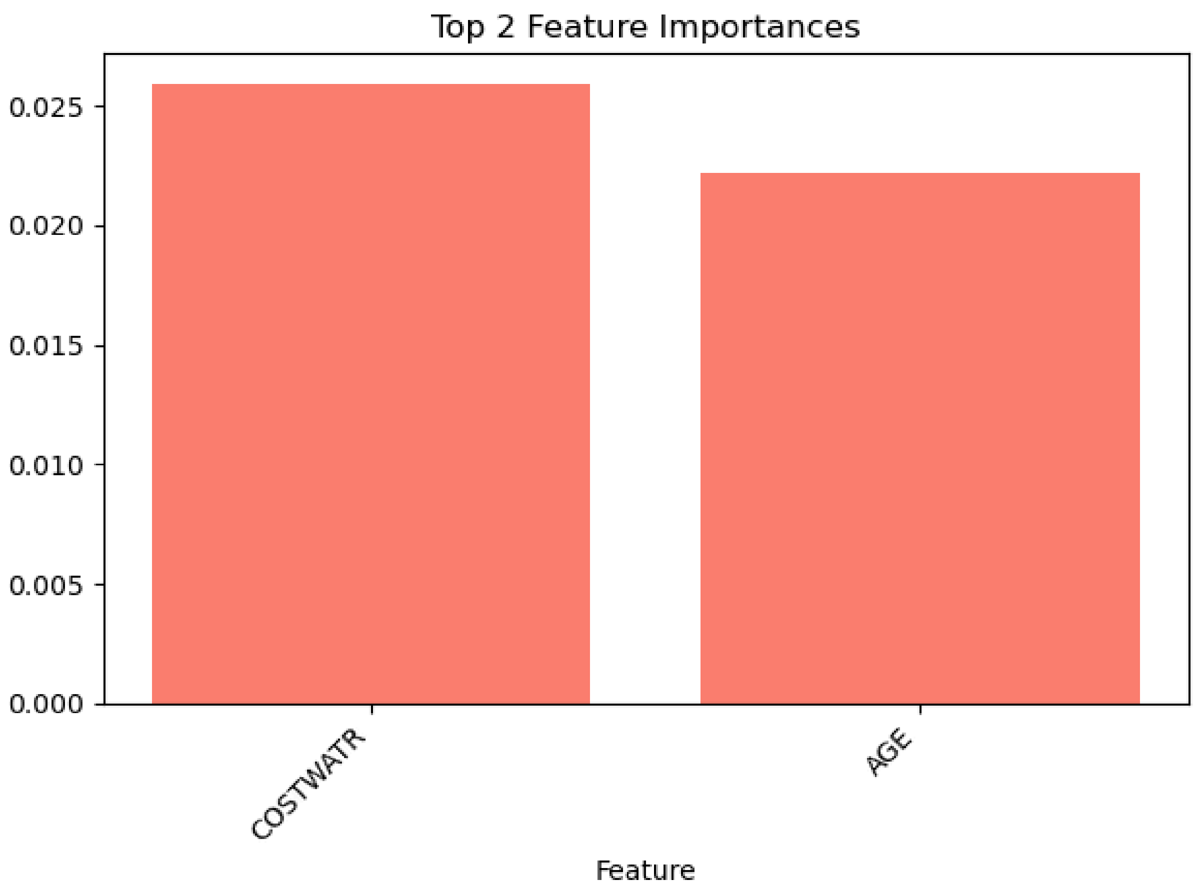
In [34]:
```
# Map importances to feature names
importances_dict = {feature: importance for feature, importance in zip(X.columns, i

# Sort features by importance in ascending order
sorted_importances = sorted(importances_dict.items(), key=lambda x: x[1], reverse=T

# Display features with their importance in ascending order
print("Features with their importance (in descending order):")
for feature, importance in sorted_importances:
    print(f"{feature}: {importance}")
```

```
Features with their importance (in descending order):
COSTWATR: 0.02588493369229241
AGE: 0.022165719413700335
ROOMS: 0.01849636055439232
HHINCOME: 0.012005184963605609
COSTELEC: 0.011995213879748778
VEHICLES: 0.009801575431249421
COSTGAS: 0.008904177884136055
BUILTYR2: 0.004207797387576084
COSTFUEL: 0.003061122744042333
```

In [35]:
```
import matplotlib.pyplot as plt

# Extract features and importances from sorted_importances
features = [item[0] for item in sorted_importances[:5]]
importances = [item[1] for item in sorted_importances[:5]]

# Plot the bar chart with a different color
plt.bar(features, importances, color='salmon')

# Set title and labels
plt.title('Top 5 Feature Importances')
plt.xlabel('Feature')
plt.ylabel('Importance')

# Rotate x-axis labels for better readability
plt.xticks(rotation=45, ha='right')

# Remove y-axis label and grid lines
plt.ylabel('')
plt.grid(False)

# Show plot
plt.tight_layout()
plt.show()
```



In [36]:
```python
import matplotlib.pyplot as plt

# Extract features and importances for top 2 predictors
top_features = [item[0] for item in sorted_importances[:2]]
top_importances = [item[1] for item in sorted_importances[:2]]

# Plot the bar chart with a different color
plt.bar(top_features, top_importances, color='salmon')

# Set title and labels
plt.title('Top 2 Feature Importances')
plt.xlabel('Feature')
plt.ylabel('Importance')

# Rotate x-axis labels for better readability
plt.xticks(rotation=45, ha='right')

# Remove y-axis label and grid lines
plt.ylabel('')
plt.grid(False)

# Show plot
plt.tight_layout()
plt.show()
```

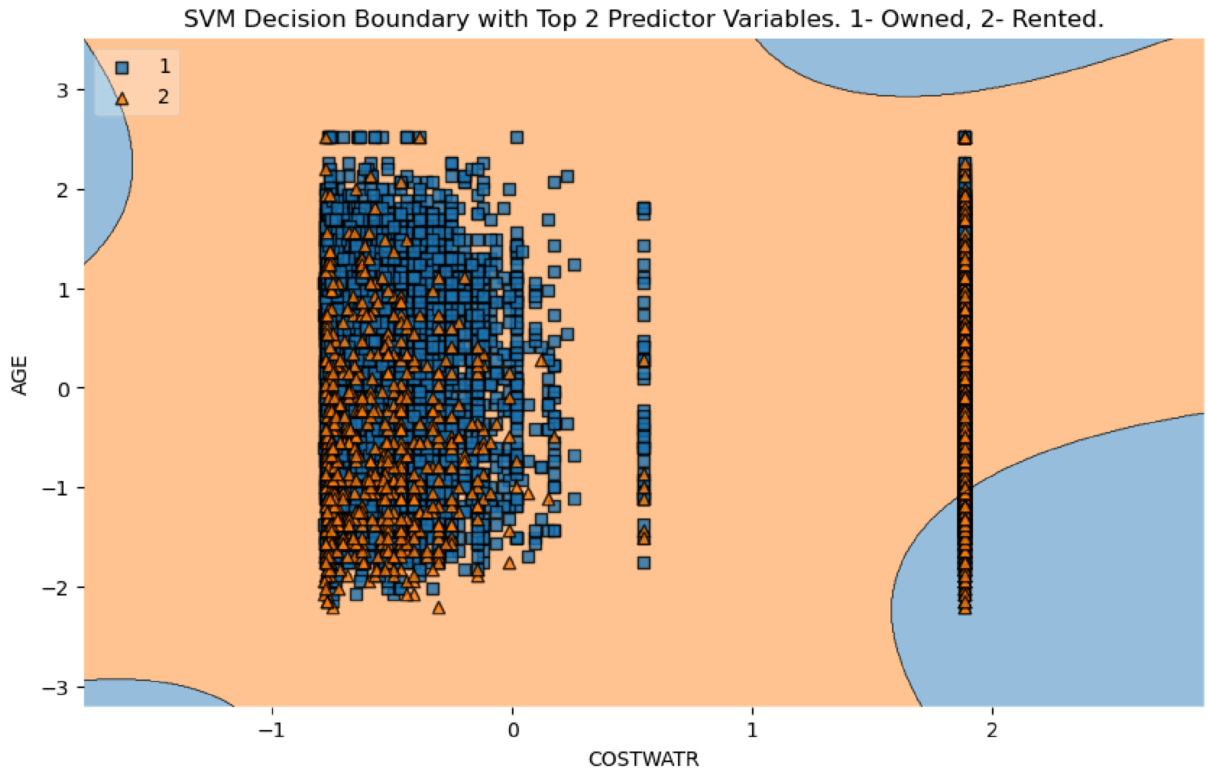


In [37]:
```
top_features
```

Out[37]:
```
['COSTWATR', 'AGE']
```

In [38]:
```
from mlxtend.plotting import plot_decision_regions
# Filter the dataset to keep only the top 2 predictor variables
X_new = X[top_features]

# Split the data into training and testing sets
X_train_2, X_test_2, y_train_2, y_test_2 = train_test_split(X_new, y, test_size=0.2
# Scale the training and testing data
scaler = StandardScaler()
X_train_2_n = scaler.fit_transform(X_train_2)
X_test_2_n = scaler.transform(X_test_2)
# Train the best model with the top 2 predictor variables
poly_svc_2 = SVC(kernel='poly',degree=4, C= 1, cache_size=1000, verbose=True, max_i
poly_svc_2.fit(X_train_2_n, y_train_2)

# Plot the decision boundary
plt.figure(figsize=(10, 6))
plot_decision_regions(X_test_2_n, y_test_2.to_numpy(), clf=poly_svc_2, legend=2)
plt.xlabel(top_features[0])
plt.ylabel(top_features[1])
plt.title('SVM Decision Boundary with Top 2 Predictor Variables. 1- Owned, 2- Rente
plt.show()
```

```
[LibSVM]
```



RADIAL KERNEL

In [39]:

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

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta

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

In [40]:

```
C_values = [0.01, 0.1, 1, 10, 100]
gamma_values = [0.1, 1, 10]

for i in C_values:
    for gamma in gamma_values:
        radial_svc = SVC(kernel='rbf', gamma=gamma, C=i, cache_size=1000, verbose=T

        radial_svc.fit(X_train_scaled, y_train)

        accuracy = radial_svc.score(X_test_scaled, y_test)

        print(f"Radial SVM with C={i} and Gamma = {gamma}: {accuracy * 100:.2f}%")
```

```
[LibSVM]Radial SVM with C=0.01 and Gamma = 0.1: 44.52%
[LibSVM]Radial SVM with C=0.01 and Gamma = 1: 19.62%
[LibSVM]Radial SVM with C=0.01 and Gamma = 10: 27.93%
[LibSVM]Radial SVM with C=0.1 and Gamma = 0.1: 49.23%
[LibSVM]Radial SVM with C=0.1 and Gamma = 1: 31.65%
[LibSVM]Radial SVM with C=0.1 and Gamma = 10: 87.32%
[LibSVM]Radial SVM with C=1 and Gamma = 0.1: 69.53%
[LibSVM]Radial SVM with C=1 and Gamma = 1: 85.63%
[LibSVM]Radial SVM with C=1 and Gamma = 10: 87.65%
[LibSVM]Radial SVM with C=10 and Gamma = 0.1: 43.57%
[LibSVM]Radial SVM with C=10 and Gamma = 1: 84.18%
[LibSVM]Radial SVM with C=10 and Gamma = 10: 85.59%
[LibSVM]Radial SVM with C=100 and Gamma = 0.1: 64.22%
[LibSVM]Radial SVM with C=100 and Gamma = 1: 83.06%
[LibSVM]Radial SVM with C=100 and Gamma = 10: 84.91%
```

In [41]:
```
estimator = RandomForestClassifier(random_state=42)
selector = SelectFromModel(estimator, threshold=-np.inf, max_features=9) # Perform
selector.fit(X_train_scaled, y_train)
X_train_new = selector.transform(X_train_scaled) # Transform the training and testi
X_test_new = selector.transform(X_test_scaled)
```

In [42]:
```
C_values = [0.01, 0.1, 1, 10, 100]
gamma_values = [0.1, 1, 10]
for i in C_values:
    for gamma in gamma_values:
        radial_svc = SVC(kernel='rbf', gamma=gamma, C=i, cache_size=1000, verbose=T
        radial_svc.fit(X_train_new, y_train)
        accuracy = radial_svc.score(X_test_new, y_test)
        print(f"Radial SVM with C={i} and Gamma = {gamma}: {accuracy * 100:.2f}%")
```

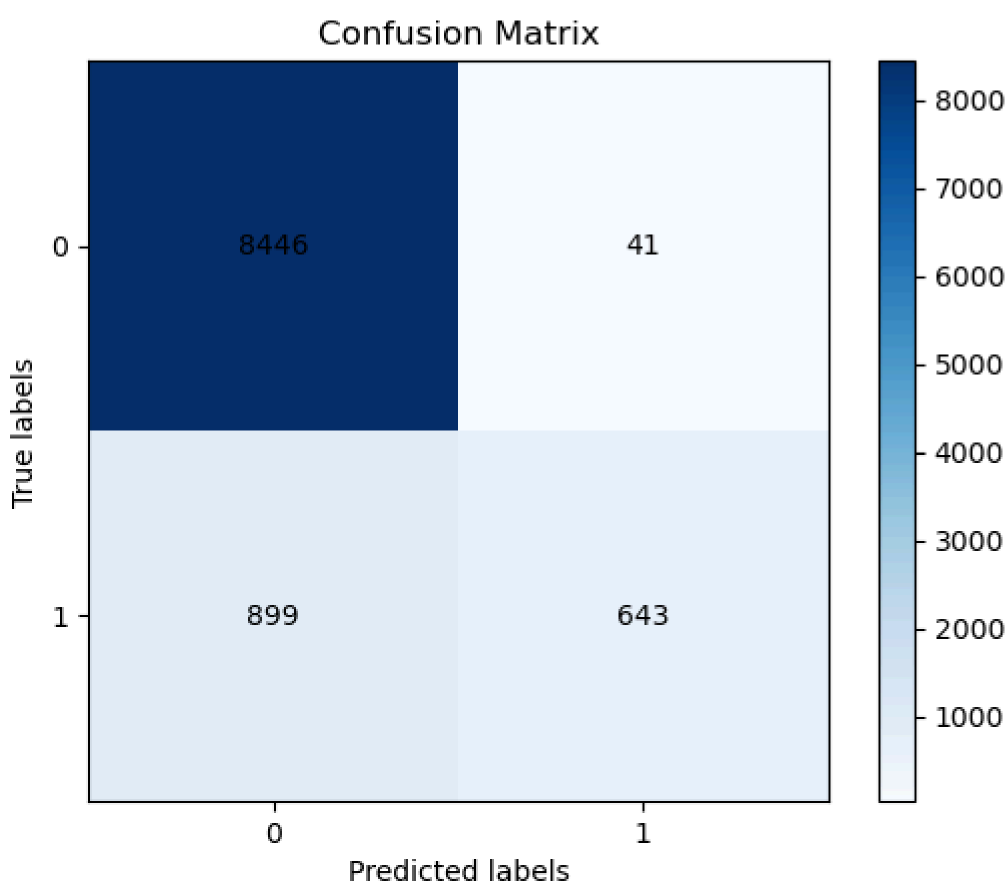
```
[LibSVM]Radial SVM with C=0.01 and Gamma = 0.1: 44.52%
[LibSVM]Radial SVM with C=0.01 and Gamma = 1: 19.62%
[LibSVM]Radial SVM with C=0.01 and Gamma = 10: 27.93%
[LibSVM]Radial SVM with C=0.1 and Gamma = 0.1: 49.23%
[LibSVM]Radial SVM with C=0.1 and Gamma = 1: 31.65%
[LibSVM]Radial SVM with C=0.1 and Gamma = 10: 87.32%
[LibSVM]Radial SVM with C=1 and Gamma = 0.1: 69.53%
[LibSVM]Radial SVM with C=1 and Gamma = 1: 85.63%
[LibSVM]Radial SVM with C=1 and Gamma = 10: 87.65%
[LibSVM]Radial SVM with C=10 and Gamma = 0.1: 43.57%
[LibSVM]Radial SVM with C=10 and Gamma = 1: 84.18%
[LibSVM]Radial SVM with C=10 and Gamma = 10: 85.59%
[LibSVM]Radial SVM with C=100 and Gamma = 0.1: 64.22%
[LibSVM]Radial SVM with C=100 and Gamma = 1: 83.06%
[LibSVM]Radial SVM with C=100 and Gamma = 10: 84.91%
```

In [43]:
```
# best at g=10 and c=1
```

In [44]:
```
radial_svc_f = SVC(kernel='rbf',gamma=10, C=1, cache_size=1000, verbose = True, max
radial_svc_f.fit(X_train_new, y_train)
accuracy = radial_svc_f.score(X_test_new, y_test)
print("Linear SVM with Gamma = 10 and C=1: {:.2f}%".format(accuracy * 100))
```

```
[LibSVM]Linear SVM with Gamma = 10 and C=1: 90.63%
```

In [45]:
```
y_pred = radial_svc_f.predict(X_test_new)

# Calculating the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Plotting the confusion matrix using imshow
plt.imshow(conf_matrix, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.colorbar()
tick_marks = np.arange(len(conf_matrix))
plt.xticks(tick_marks, tick_marks)
plt.yticks(tick_marks, tick_marks)
plt.xlabel('Predicted labels')
plt.ylabel('True labels')

# Displaying the values in the cells
for i in range(len(conf_matrix)):
    for j in range(len(conf_matrix)):
        plt.text(j, i, conf_matrix[i, j], horizontalalignment='center', verticalali

plt.show()
```



In [47]:
```
from sklearn.inspection import permutation_importance
result = permutation_importance(radial_svc_f, X_test_new, y_test, n_repeats=5, rand
importances = result.importances_mean
```

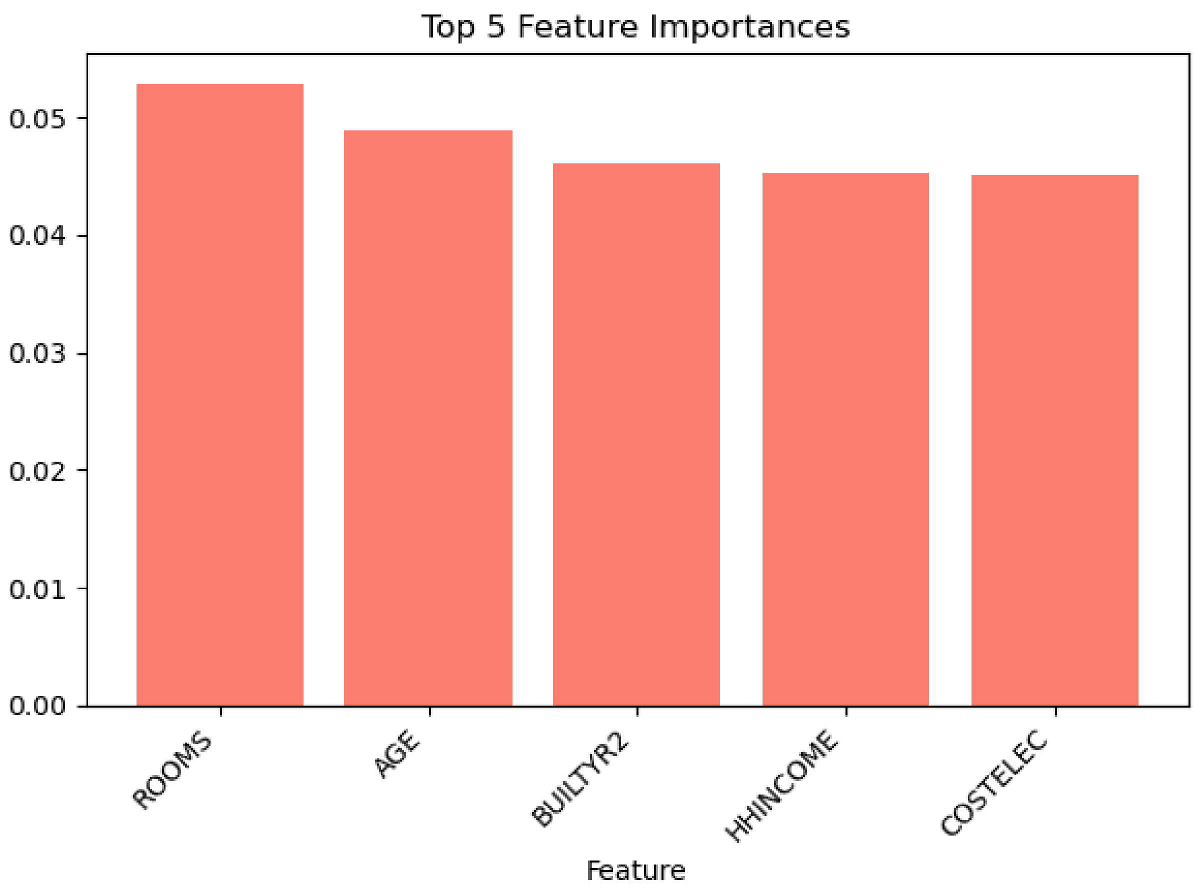
In [48]:
```
# Map importances to feature names
importances_dict = {feature: importance for feature, importance in zip(X.columns, i
```

```python
# Sort features by importance in ascending order
sorted_importances = sorted(importances_dict.items(), key=lambda x: x[1], reverse=T

# Display features with their importance in ascending order
print("Features with their importance (in descending order):")
for feature, importance in sorted_importances:
    print(f"{feature}: {importance}")
```

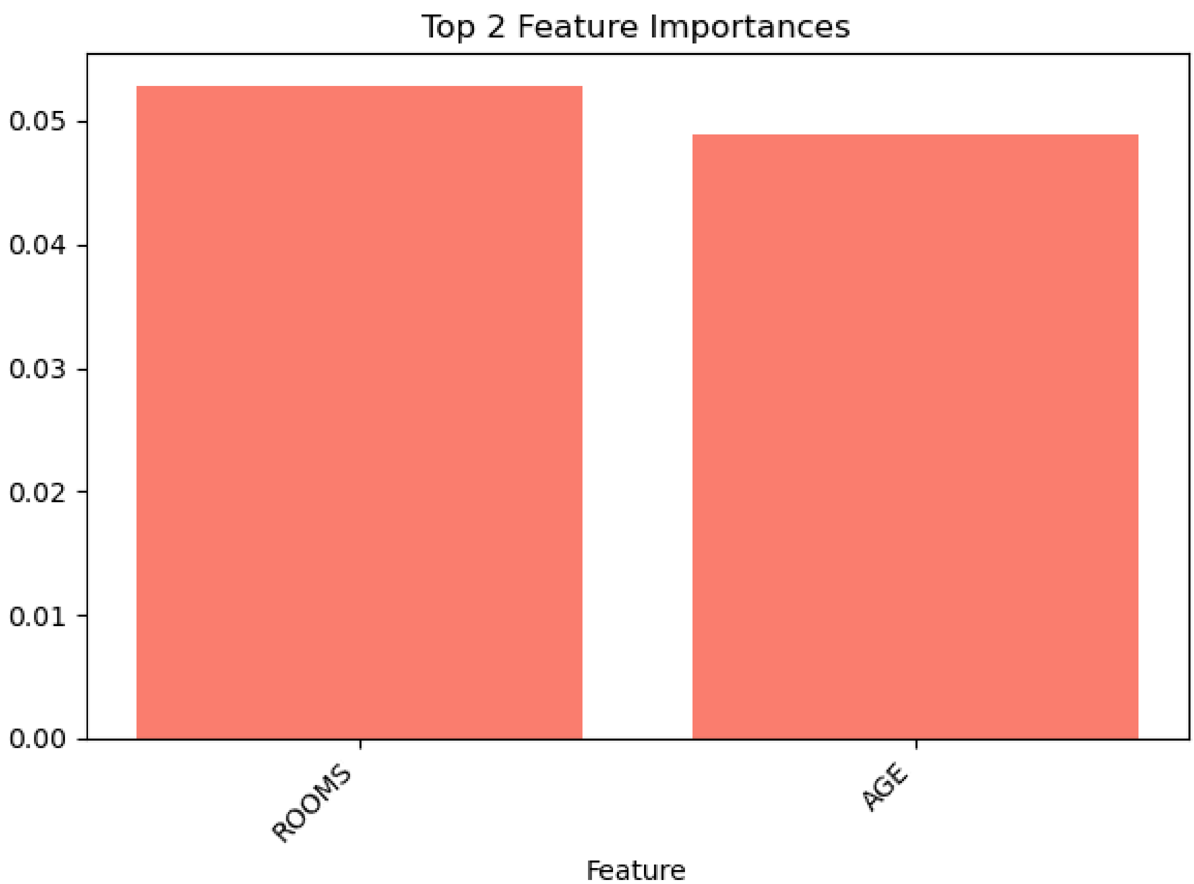
```
Features with their importance (in descending order):
ROOMS: 0.05278691793797981
AGE: 0.04889819523382186
BUILTYR2: 0.046146176089340865
HHINCOME: 0.04528866287765476
COSTELEC: 0.04514906770365932
COSTWATR: 0.042935487087446364
VEHICLES: 0.039744740253265465
COSTGAS: 0.03168810449695878
COSTFUEL: 0.009472529663974449
```

In [49]:

```python
import matplotlib.pyplot as plt

# Extract features and importances from sorted_importances
features = [item[0] for item in sorted_importances[:5]]
importances = [item[1] for item in sorted_importances[:5]]

# Plot the bar chart with a different color
plt.bar(features, importances, color='salmon')

# Set title and labels
plt.title('Top 5 Feature Importances')
plt.xlabel('Feature')
plt.ylabel('Importance')

# Rotate x-axis labels for better readability
plt.xticks(rotation=45, ha='right')

# Remove y-axis label and grid lines
plt.ylabel('')
plt.grid(False)

# Show plot
plt.tight_layout()
plt.show()
```



In [50]:
```python
import matplotlib.pyplot as plt

# Extract features and importances for top 2 predictors
top_features = [item[0] for item in sorted_importances[:2]]
top_importances = [item[1] for item in sorted_importances[:2]]

# Plot the bar chart with a different color
plt.bar(top_features, top_importances, color='salmon')

# Set title and labels
plt.title('Top 2 Feature Importances')
plt.xlabel('Feature')
plt.ylabel('Importance')

# Rotate x-axis labels for better readability
plt.xticks(rotation=45, ha='right')

# Remove y-axis label and grid lines
plt.ylabel('')
plt.grid(False)

# Show plot
plt.tight_layout()
plt.show()
```

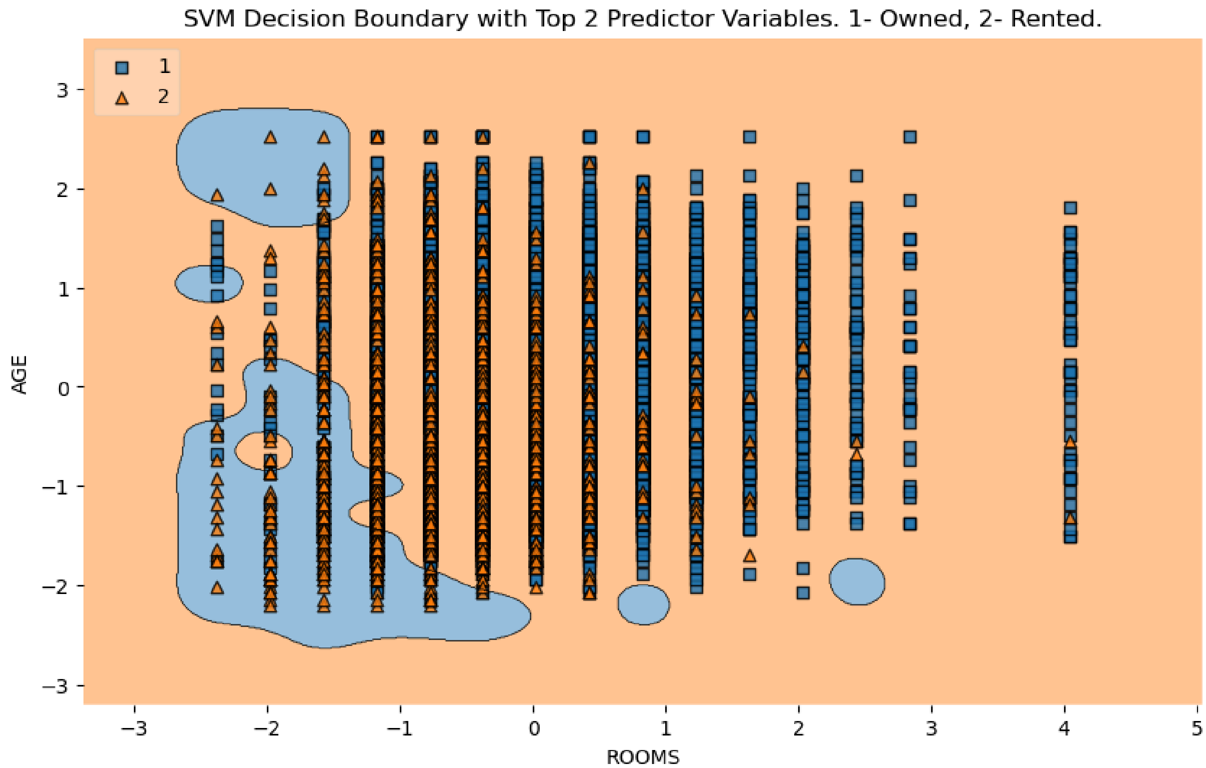


In [51]:
```
top_features
```

Out[51]:
```
['ROOMS', 'AGE']
```

In [52]:
```
from mlxtend.plotting import plot_decision_regions
# Filter the dataset to keep only the top 2 predictor variables
X_new = X[top_features]

# Split the data into training and testing sets
X_train_2, X_test_2, y_train_2, y_test_2 = train_test_split(X_new, y, test_size=0.2
# Scale the training and testing data
scaler = StandardScaler()
X_train_2_n = scaler.fit_transform(X_train_2)
X_test_2_n = scaler.transform(X_test_2)
# Train the best model with the top 2 predictor variables
radial_svc_2 = SVC(kernel='rbf',gamma=10, C= 1, cache_size=1000, verbose=True, max_
radial_svc_2.fit(X_train_2_n, y_train_2)

# Plot the decision boundary
plt.figure(figsize=(10, 6))
plot_decision_regions(X_test_2_n, y_test_2.to_numpy(), clf=radial_svc_2, legend=2)
plt.xlabel(top_features[0])
plt.ylabel(top_features[1])
plt.title('SVM Decision Boundary with Top 2 Predictor Variables. 1- Owned, 2- Rente
plt.show()
```

```
[LibSVM]
```


SVM Decision Boundary with Top 2 Predictor Variables. 1- Owned, 2- Rented.
1
2
AGE
ROOMS
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3
4
5

In [ ]:

In [ ]:

In [ ]: