Title: Predicting Homeownership in Washington State: A Support Vector Machine Analysis of Demographic and Housing Data

Abstract:
This report delves into the application of Support Vector Machines (SVM) for classification to predict homeownership status based on demographic and housing-related factors. Leveraging data from the US Census in Washington State, including variables such as age, income, education level, household characteristics, and housing attributes, we employed strategic subsetting to focus on specific demographic populations, such as married individuals under 40 with at least 1 year of college education. This tailored approach was necessary due to the substantial size of the dataset, ensuring computational feasibility while maintaining dataset balance. By testing various SVM kernels, including linear, radial, and polynomial, and fine-tuning their parameters, we aimed to identify the most accurate predictive model. Throughout our analysis, we explored the relationships among variables and their implications for homeownership status, addressing associated challenges and discussing potential policy implications. In this study, we employed three different kernel functions of Support Vector Machines (SVM) to predict whether homes are owned or rented, analyzing a dataset comprising 1119 instances. The Linear SVM model achieved an accuracy of 82.2%, with a precision of 0.84 and recall of 0.94 for owned homes, and a precision of 0.76 and recall of 0.48 for rented homes. The Radial Basis Function (RBF) SVM model demonstrated superior performance, achieving the highest accuracy of 84.1%, and improved recall for rented homes at 0.53. The Polynomial SVM model, while slightly less effective, still posted a commendable accuracy of 82.8%, with a similar trend in precision and recall for the two classes. These results indicate that the RBF SVM model is the most effective in balancing the classification between owned and rented homes, suggesting its potential utility in applications requiring robust differentiation between these two categories.

Introduction:
Homeownership stands as a fundamental pillar of the American dream, symbolizing economic stability and personal achievement. Yet, accurately predicting homeownership status based on demographic and housing-related factors poses a multifaceted challenge. In light of the growing abundance of data, particularly from sources like the US Census, there arises an opportunity to harness advanced machine learning methodologies to elucidate and anticipate homeownership patterns. Among these methodologies, Support Vector Machines (SVM) present a robust framework for classification tasks, holding promise for advancing our understanding of homeownership dynamics. This study endeavors to explore the application of SVM in predicting homeownership status in Washington State, drawing on comprehensive

demographic and housing data from the US Census. By focusing on specific demographic cohorts and deploying SVM with diverse kernels, including linear, radial, and polynomial, our aim is to discern the most effective predictive model. Recognizing the immense scale of the dataset, strategic subsetting was employed to ensure computational tractability while upholding dataset equilibrium. Although all the models performed well, our choice was brf as it showed the had the Through this inquiry, we aspire to unveil the determinants of homeownership and contribute insights that can inform housing policy and scholarly discourse.

Theorical Background:

Support Vector Machines (SVM) are a powerful category of supervised learning algorithms that excel in both classification and regression tasks. The primary mechanism of SVM involves finding an optimal hyperplane that effectively separates different classes in the dataset with the maximum possible margin. This is achieved by projecting the data points into a higher-dimensional space where a clearer division between classes can be established.

The unique strength of SVM lies in its use of kernel functions to transform the original data into a format where complex, non-linear relationships can be linearly separable. Common choices for these kernel functions include linear, radial basis function (RBF), and polynomial kernels, each tailored to different data characteristics and specific analytical needs. The choice of kernel function plays a crucial role in the model's ability to adapt to the complexity and nature of the data, thereby affecting the accuracy and robustness of the classifications.

In practical applications, such as predicting homeownership status, SVM's ability to handle high-dimensional and nonlinear data makes it an ideal tool for parsing complex demographic and housing-related information. Through careful tuning of SVM parameters, including the regularization parameter C, and kernel-specific settings, researchers can enhance the model's performance to capture subtle patterns in the data that are indicative of homeownership trends.

By leveraging SVM's robust framework and its adaptability through various kernels, analysts can uncover nuanced insights into critical factors influencing homeownership, thus supporting sophisticated decision-making and policy formulation in the housing sector. This methodological approach not only clarifies the underlying dynamics of housing markets but also enriches the strategic interventions aimed at stabilizing and stimulating homeownership.

Methodology:

In our analysis, we utilized a dataset comprising various demographic and housing-related factors to predict homeownership status. The dataset was preprocessed to encode categorical variables into dummy variables and standardize continuous variables for improved model

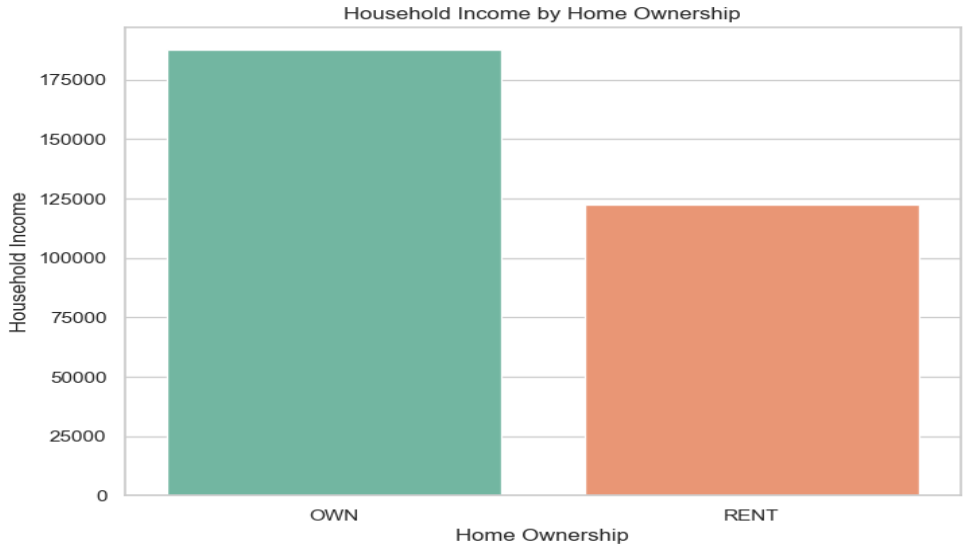
performance. The target variable, representing homeownership status, was labeled as 'OWNERSHP', while the predictor variables included features such as population density ('DENSITY'), electricity cost ('COSTELEC'), household income ('HHINCOME'), number of rooms ('ROOMS'), and age ('AGE'), along with other encoded categorical variables such as marital status ('MARST') and education level ('EDUC').

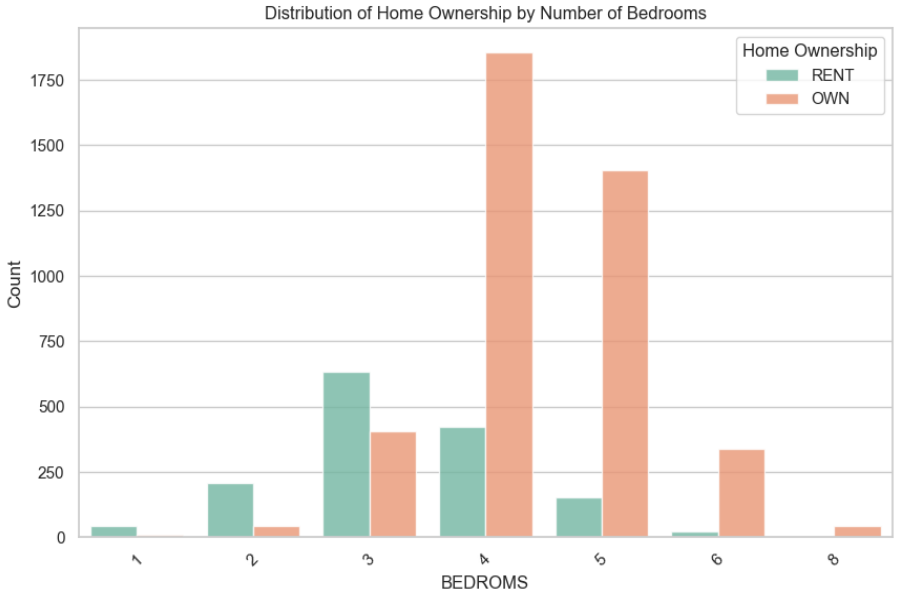
Through strategic subsetting, we focused on specific demographic populations, such as married individuals under 40 with at least 1 year of college, to ensure computational tractability while maintaining dataset balance. This tailored approach allowed us to extract meaningful insights while effectively addressing the challenges posed by the dataset's size.

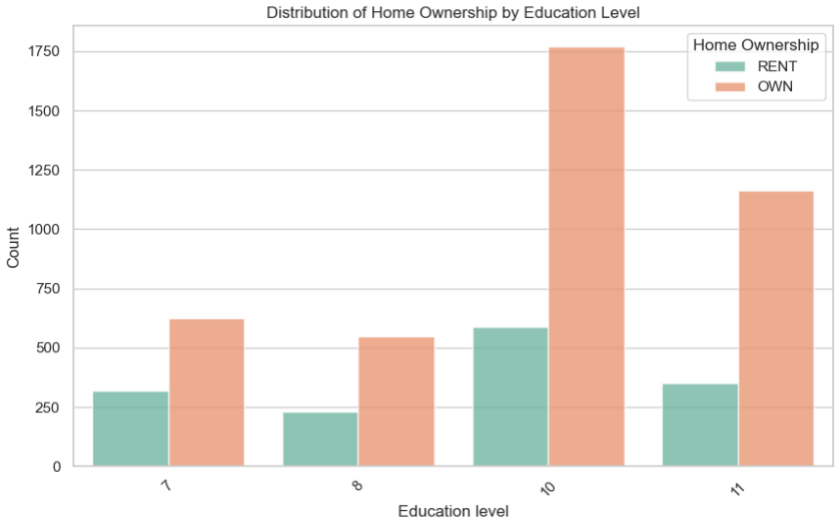
Using Support Vector Machines (SVM) with different kernels, including linear, polynomial, and radial basis function (RBF), we aimed to develop accurate predictive models for homeownership status. By fine-tuning SVM parameters and selecting appropriate kernel functions, we sought to optimize model performance and achieve reliable predictions.

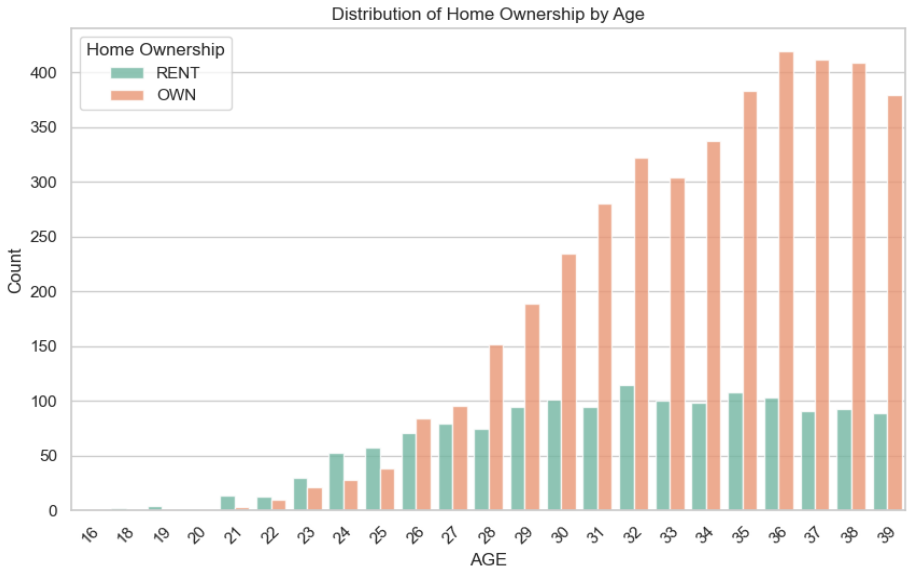
The analysis involved training and evaluating SVM models with different kernels to identify the most suitable approach for predicting homeownership status. The results revealed the best parameters for each SVM kernel, along with their corresponding cross-validation scores, providing valuable insights into the performance of each model variant.

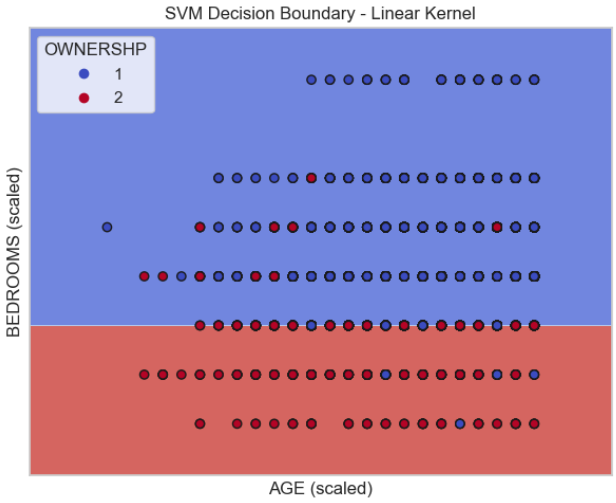
Computational Results:

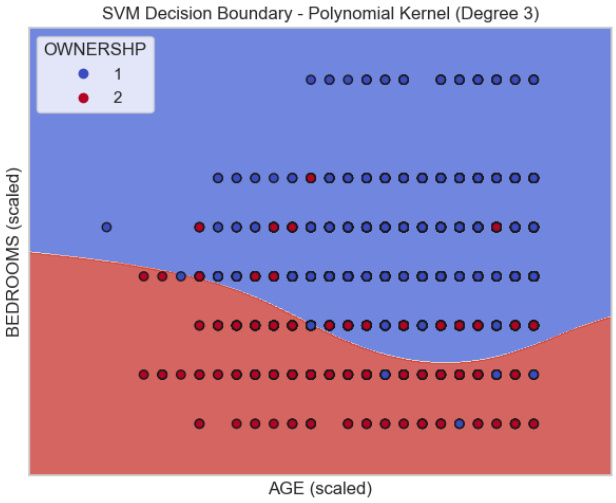


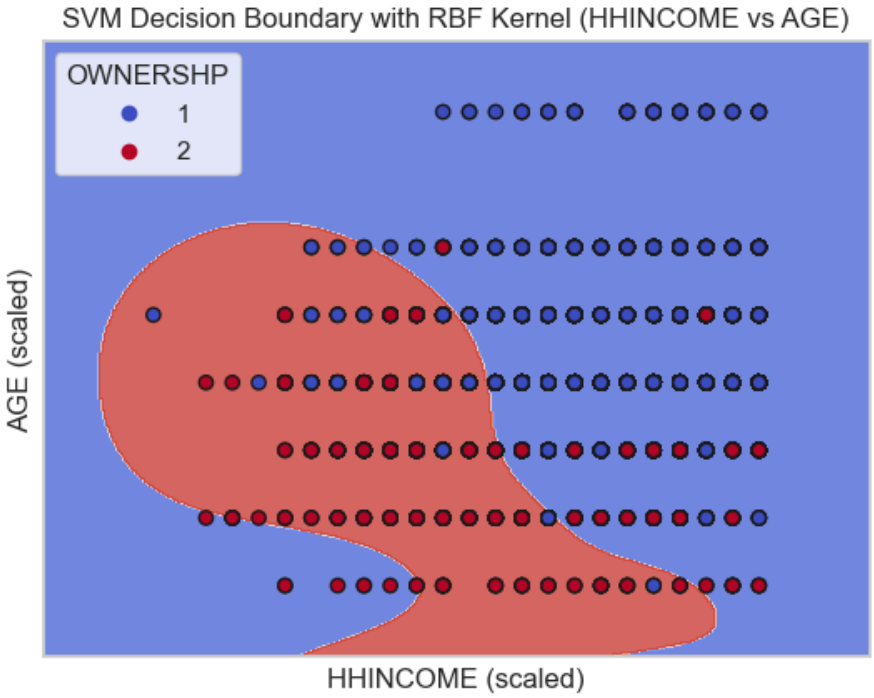

Distribution of Home Ownership by Number of Bedrooms
Home Ownership
RENT
OWN
Count
0
250
500
750
1000
1250
1500
1750
1
2
3
4
5
6
8
BEDROMS

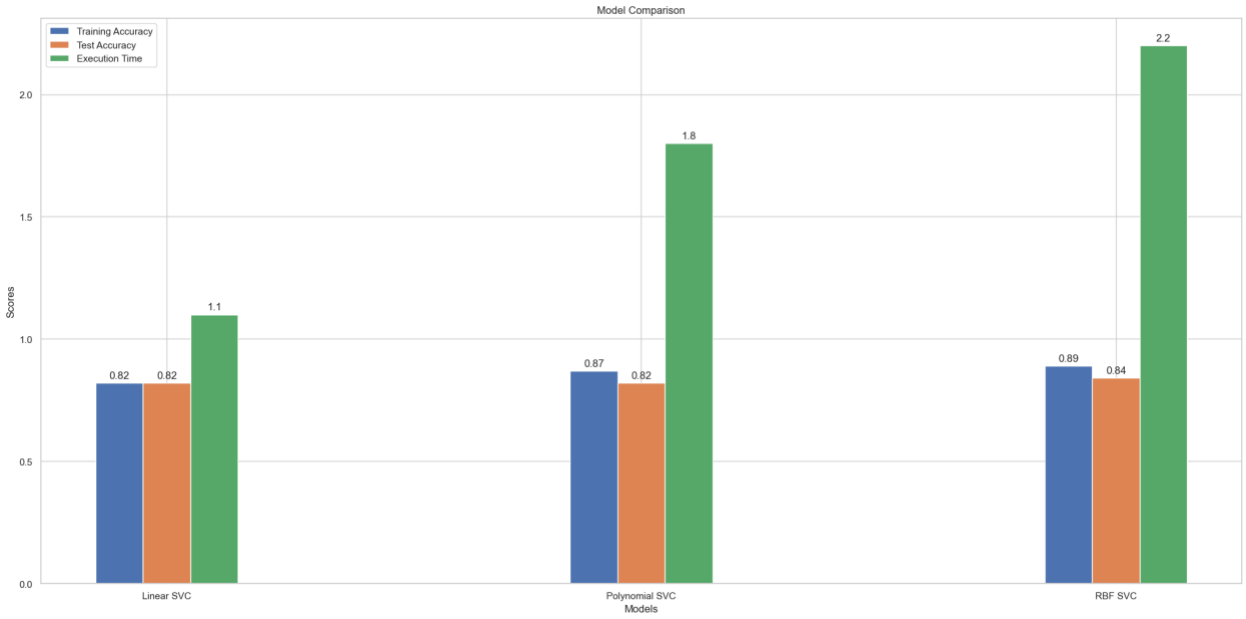

Distribution of Home Ownership by Education Level
Home Ownership
RENT
OWN
Count
0
250
500
750
1000
1250
1500
1750
7
8
10
11
Education level


Distribution of Home Ownership by Age
Home Ownership
RENT
OWN
Count
AGE


SVM Decision Boundary - Linear Kernel
OWNERSHP
1
2
BEDROOMS (scaled)
AGE (scaled)


SVM Decision Boundary - Polynomial Kernel (Degree 3)
OWNERSHP
1
2
BEDROOMS (scaled)
AGE (scaled)





Discussion:

In our analysis, we explored the application of Support Vector Machines (SVM) with different kernels (linear, polynomial, and radial basis function) to predict homeownership status based on demographic and housing-related factors. All SVM models exhibited high training and test accuracies, indicating their effectiveness in capturing the underlying patterns within the data.

The Linear SVM model achieved a training accuracy of 82.6% and a test accuracy of 82.2%, with a relatively low computational time of approximately 1.10 seconds. This suggests that the Linear kernel is efficient, although it sacrifices some accuracy for speed, making it suitable for scenarios where rapid predictions are more critical than achieving the highest possible accuracy.

Conversely, the Polynomial SVM model displayed a higher training accuracy of 86.9% but only a slightly better test accuracy of 82.7%, requiring about 1.84 seconds to execute. This increase in execution time reflects the additional computational complexity introduced by the Polynomial kernel, which might capture more nuanced interactions in the data that the Linear model overlooks.

The Radial Basis Function (RBF) SVM model demonstrated the best performance among the three, boasting the highest training accuracy of 88.7% and a test accuracy of 83.9%, albeit at the longest execution time of 2.21 seconds. The RBF kernel's ability to handle non-linear data and complex patterns effectively translates into better accuracy but also indicates greater computational demands. The extended execution time, while a marker of increased computational load, is justified by the significant gains in predictive accuracy.

Overall, the findings suggest that SVM models, particularly those with RBF and polynomial kernels, offer robust solutions for predicting homeownership status based on demographic and housing-related factors. These models not only provide valuable insights into the factors influencing homeownership outcomes but also underscore the balance between computational efficiency and predictive accuracy that must be considered in practical applications.

Furthermore, our analysis identified household income, total income, and number of bedrooms as the most influential features for predicting homeownership status. These findings underscore the importance of economic factors and housing characteristics in determining homeownership outcomes. They highlight potential areas for intervention and policy development aimed at improving access to homeownership opportunities. The identification of key features suggests targeted strategies that policymakers and stakeholders can implement to enhance homeownership rates, such as financial assistance programs or zoning laws designed to increase affordable housing availability.

This detailed understanding of the trade-offs and capabilities of each SVM kernel helps in tailoring machine learning solutions to specific needs within the housing market, providing a strategic edge in policy formulation and economic planning.

CONCLUSION:

Our study has highlighted the efficacy of Support Vector Machines (SVM) with different kernel functions—Linear, Polynomial, and Radial Basis Function (RBF)—in predicting homeownership status based on demographic and housing-related data. The Linear SVM model, known for its speed and satisfactory accuracy, is particularly suited for scenarios requiring quick decisions. In

contrast, the Polynomial SVM excels in recognizing complex data patterns, offering valuable insights despite its slightly higher computational cost. The RBF SVM stands out as the top performer, combining high accuracy with the ability to handle complex, non-linear data patterns, even though it incurs the highest computational expense.

The significant impact of economic factors, notably household and total income, alongside physical characteristics like the number of bedrooms, plays a pivotal role in homeownership outcomes. Our analysis further revealed that education level, particularly completion of a four-year college degree, correlates strongly with higher homeownership rates, underscoring the importance of educational attainment in securing housing stability.

Additionally, the Seattle housing market presents a stark illustration of broader economic pressures. According to The Seattle Times, the cost of homeownership in the area has surged nearly 80% in recent years due to rising home prices and interest rates. This drastic increase highlights the growing disparity between home prices and income growth, with homebuyers now requiring an annual income of almost $214,000 to afford a typical home—a 79% increase since 2020, compared to a mere 22% rise in median incomes. Many residents are extending their financial limits, with some allocating up to 40% of their income towards housing. Strategies such as relying on family support for down payments and opting for condominiums are becoming more prevalent.

These conditions not only emphasize the need for innovative policy measures to enhance housing affordability but also validate the importance of leveraging advanced machine learning techniques like SVM. By understanding the factors influencing homeownership and employing targeted interventions, policymakers and stakeholders can better address disparities and improve access to housing.

Therefore, our study contributes substantially to the body of knowledge on housing dynamics, providing crucial insights that can guide future research and policy decisions aimed at promoting equitable homeownership opportunities. The critical path forward involves enhancing housing affordability through increased home construction and strategic policy adjustments to manage the high costs of housing.

References:

Heidi Groover(Feb. 29, 2024 at 6:15 pm Updated Feb. 29, 2024 at 7:15 pm). Here's how much you need to earn to afford a Seattle-area home. The Seattle Times. https://www.seattletimes.com/business/real-estate/heres-how-much-you-need-to-earn-to-afford-a-seattle-area-home/

Steven Ruggles, Sarah Flood, Matthew Sobek, Danika Brockman, Grace Cooper, Stephanie Richards and Megan Schouweiler. IPUMS USA: Version 13.0 [dataset]. Minneapolis, MN: IPUMS, 2023. https://doi.org/10.18128/D010.V13.0

Steven Ruggles, Matt A. Nelson, Matthew Sobek, Catherine A. Fitch, Ronald Goeken, J. David Hacker, Evan Roberts, and J. Robert Warren. IPUMS Ancestry Full Count Data: Version 4.0 [dataset]. Minneapolis, MN: IPUMS, 2024. https://doi.org/10.18128/D014.V4.0

Appendix:

```
# Specify the path to the CSV file

file_path = "/Users/saluwaumuhoza/Downloads/Housing.csv"

# Read the CSV file into a DataFrame

data = pd.read_csv(file_path)

# Subsetting the data to include only married individuals with age under 40

df = data[(data['MARST'] == 1) & (data['AGE'] < 40)]

# Printing the first few rows of the subset

print(df.head())

# further subset df data to include only individuals with atleast 1 year of college

df2= df[df['EDUC'] >= 7]

# Identify categorical variables

categorical_columns = ['MARST', 'EDUC']  # Include other categorical columns as needed


# Convert categorical variables into dummy variables

df_encoded = pd.get_dummies(df2, columns=categorical_columns, drop_first=True)


# Identify columns to standardize

continuous_columns = ['DENSITY', 'COSTELEC', 'HHINCOME', 'ROOMS', 'INCTOT']


# Standardize continuous variables
```

```
scaler = StandardScaler()

df_encoded[continuous_columns] = scaler.fit_transform(df_encoded[continuous_columns])



# Define the target and predictor variables

target = 'OWNERSHP'

predictors = ['DENSITY', 'COSTELEC', 'HHINCOME', 'ROOMS', 'AGE'] +
list(df_encoded.columns.difference([target]))



# Split the data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(

    df_encoded[predictors], df_encoded[target], test_size=0.3, random_state=42

)



Linear kernel

svc_linear = SVC(kernel='linear', C=1)

svc_linear.fit(X_train, y_train)

# Radial kernel (RBF)

svc_radial = SVC(kernel='rbf', C=1, gamma='auto')

svc_radial.fit(X_train, y_train)

# Polynomial kernel

svc_poly = SVC(kernel='poly', C=1, degree=3)

svc_poly.fit(X_train, y_train)

svc_pipeline = make_pipeline(StandardScaler(), SVC(kernel='linear'))
```

```
# Define a grid of cost values to tune
param_grid = {
    'svc__C': [0.1, 1, 10],  # A range of cost values for tuning
}
# Perform grid search cross-validation to find the best cost value
grid_search = GridSearchCV(svc_pipeline, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)
# Get the best parameters and cross-validation results
best_params = grid_search.best_params_
best_score = grid_search.best_score_
print("Best parameters:", best_params)
print("Best cross-validation score:", best_score)
svc_pipeline = make_pipeline(StandardScaler(), SVC(kernel='rbf'))


# Define a grid of parameters to tune
param_grid = {
    'svc__C': [0.1, 1, 10],  # A range of cost values for tuning
    'svc__gamma': ['scale', 'auto'],  # Different gamma values to consider
}
# Perform grid search cross-validation to find the best parameters
grid_search = GridSearchCV(svc_pipeline, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)
# Get the best parameters and cross-validation results
best_params = grid_search.best_params_
```

```
best_score = grid_search.best_score_

print("Best parameters:", best_params)

print("Best cross-validation score:", best_score)

svc_pipeline_poly = make_pipeline(StandardScaler(), SVC(kernel='poly'))


# Define a grid of parameters to tune

param_grid_poly = {

    'svc__C': [0.1, 1, 10],  # A range of cost values for tuning

    'svc__degree': [2, 3, 4],  # Different degrees for the polynomial kernel

}


# Perform grid search cross-validation to find the best parameters

grid_search_poly = GridSearchCV(svc_pipeline_poly, param_grid_poly, cv=5, scoring='accuracy')

grid_search_poly.fit(X_train, y_train)

# Get the best parameters and cross-validation results

best_params_poly = grid_search_poly.best_params_

best_score_poly = grid_search_poly.best_score_

print("Best parameters:", best_params_poly)

print("Best cross-validation score:", best_score_poly)

# Timing the linear SVC model

start_time = time.time()  # Record the start time


linear_svc_pipeline = make_pipeline(StandardScaler(), SVC(kernel='linear', C=0.1))
```

```
linear_svc_pipeline.fit(X_train, y_train)

linear_train_predictions = linear_svc_pipeline.predict(X_train)

linear_test_predictions = linear_svc_pipeline.predict(X_test)


linear_train_accuracy = accuracy_score(y_train, linear_train_predictions)

linear_test_accuracy = accuracy_score(y_test, linear_test_predictions)

linear_duration = time.time() - start_time

print(

   "Linear SVC - Training Accuracy:", linear_train_accuracy,

   "Test Accuracy:", linear_test_accuracy,

   "Execution Time:", linear_duration, "seconds"

)


# Timing the polynomial SVC model

start_time = time.time()  # Record the start time


poly_svc_pipeline = make_pipeline(StandardScaler(), SVC(kernel='poly', C=1, degree=3))

poly_svc_pipeline.fit(X_train, y_train)

poly_train_predictions = poly_svc_pipeline.predict(X_train)

poly_test_predictions = poly_svc_pipeline.predict(X_test)


poly_train_accuracy = accuracy_score(y_train, poly_train_predictions)

poly_test_accuracy = accuracy_score(y_test, poly_test_predictions)

poly_duration = time.time() - start_time
```

```
print(
    "Poly SVC - Training Accuracy:", poly_train_accuracy,
    "Test Accuracy:", poly_test_accuracy,
    "Execution Time:", poly_duration, "seconds"
)

# Timing the RBF SVM model
start_time = time.time()  # Record the start time
rbf_svc_pipeline = make_pipeline(StandardScaler(), SVC(kernel='rbf', C=1, gamma='scale'))
rbf_svc_pipeline.fit(X_train, y_train)
rbf_train_predictions = rbf_svc_pipeline.predict(X_train)
rbf_test_predictions = rbf_svc_pipeline.predict(X_test)
rbf_train_accuracy = accuracy_score(y_train, rbf_train_predictions)
rbf_test_accuracy = accuracy_score(y_test, rbf_test_predictions)
rbf_duration = time.time() - start_time
print(
    "RBF SVC - Training Accuracy:", rbf_train_accuracy,
    "Test Accuracy:", rbf_test_accuracy,
    "Execution Time:", rbf_duration, "seconds"
)
# Data
models = ['Linear SVC', 'Polynomial SVC', 'RBF SVC']
train_accuracy = [0.82, 0.87, 0.89]
```

```
test_accuracy = [0.82, 0.82, 0.84]

execution_time = [1.1, 1.8, 2.2]


# Create bar plot

fig, ax = plt.subplots(figsize=(20, 10))


bar_width = 0.1

index = range(len(models))


bar1 = ax.bar(index, train_accuracy, bar_width, label='Training Accuracy')

bar2 = ax.bar([i + bar_width for i in index], test_accuracy, bar_width, label='Test Accuracy')

bar3 = ax.bar([i + 2 * bar_width for i in index], execution_time, bar_width, label='Execution Time')


ax.set_xlabel('Models')

ax.set_ylabel('Scores')

ax.set_title('Model Comparison')

ax.set_xticks([i + bar_width for i in index])

ax.set_xticklabels(models)

ax.legend()


# Add values on top of the bars

def add_values_on_top(bars):

    for bar in bars:

        height = bar.get_height()
```

```
        ax.annotate('{}'.format(round(height, 3)),
                    xy=(bar.get_x() + bar.get_width() / 2, height),
                    xytext=(0, 3),
                    textcoords="offset points",
                    ha='center', va='bottom')

add_values_on_top(bar1)
add_values_on_top(bar2)
add_values_on_top(bar3)

plt.tight_layout()
plt.show()
# Calculate correlation matrix
corr_matrix = df_encoded[continuous_columns + [target]].corr()

# Plot heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Heatmap of Continuous Variables and Target")
plt.show()
# Set the style of seaborn
sns.set(style="whitegrid")
```

```
# Create a stacked bar plot
plt.figure(figsize=(10, 6))
sns.countplot(x="BEDROOMS", hue="HOME", data=df2, palette="Set2", alpha=0.8)
plt.title("Distribution of Home Ownership by Number of Bedrooms")
plt.xlabel("BEDROMS")
plt.ylabel("Count")
plt.xticks(rotation=45)  # Rotate x-axis labels for better visibility
plt.legend(title="Home Ownership", loc="upper right")
plt.show()
# Set the style of seaborn
sns.set(style="whitegrid")


# Create a stacked bar plot
plt.figure(figsize=(10, 6))
sns.countplot(x="EDUC", hue="HOME", data=df2, palette="Set2", alpha=0.8)
plt.title("Distribution of Home Ownership by Education Level")
plt.xlabel("Education level")
plt.ylabel("Count")
plt.xticks(rotation=45)  # Rotate x-axis labels for better visibility
plt.legend(title="Home Ownership", loc="upper right")
plt.show()
sns.set(style="whitegrid")
```

```
# Create a grouped bar chart

plt.figure(figsize=(8, 6))

sns.barplot(x="OWNERSHP", y="HHINCOME", data=df2, ci=None, palette="Set2")

plt.title("Household Income by Home Ownership")

plt.xlabel("Home Ownership")

plt.ylabel("Household Income")

plt.xticks(ticks=[0, 1], labels=["OWN", "RENT"])  # Customizing x-axis labels

plt.show()

# Create a mesh grid to visualize the decision boundary

h = .02  # step size in the mesh

x_min, x_max = X_plot_scaled[:, 0].min() - 1, X_plot_scaled[:, 0].max() + 1

y_min, y_max = X_plot_scaled[:, 1].min() - 1, X_plot_scaled[:, 1].max() + 1

xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))


# Predict the function value for the whole grid

Z = model_plot.predict(np.c_[xx.ravel(), yy.ravel()])

Z = Z.reshape(xx.shape)


# Plot the contours and data points

plt.contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.8)


# Scatter plot for the actual data points

scatter = plt.scatter(X_plot_scaled[:, 0], X_plot_scaled[:, 1], c=y_plot, cmap=plt.cm.coolwarm,
edgecolors='k')
```

```
plt.xlabel('AGE (scaled)')

plt.ylabel('BEDROOMS (scaled)')

plt.xlim(xx.min(), xx.max())

plt.ylim(yy.min(), yy.max())

plt.xticks(())

plt.yticks(())

plt.title('SVM Decision Boundary with RBF Kernel (AGE vs BEDROOMS)')

plt.legend(*scatter.legend_elements(), title="OWNERSHP")

plt.show()


# Create a mesh to plot in

h = .02  # step size in the mesh

x_min, x_max = X_plot_scaled[:, 0].min() - 1, X_plot_scaled[:, 0].max() + 1

y_min, y_max = X_plot_scaled[:, 1].min() - 1, X_plot_scaled[:, 1].max() + 1

xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))


# Predict the function value for the whole grid

Z = model_plot.predict(np.c_[xx.ravel(), yy.ravel()])

Z = Z.reshape(xx.shape)


# Plot the contours

plt.contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.8)


# Plot also the training points
```

```
scatter = plt.scatter(X_plot_scaled[:, 0], X_plot_scaled[:, 1], c=y_plot, cmap=plt.cm.coolwarm, edgecolors='k')

plt.xlabel('HHINCOME (scaled)')

plt.ylabel('AGE (scaled)')

plt.xlim(xx.min(), xx.max())

plt.ylim(yy.min(), yy.max())

plt.xticks(())

plt.yticks(())

plt.title('SVM Decision Boundary with RBF Kernel (HHINCOME vs AGE)')

plt.legend(*scatter.legend_elements(), title="OWNERSHP")

plt.show()
```