

This notebook contains a collection of machine learning experiments organised by lab sessions.

Lab 1: Import necessary libraries (scikit-learn, numpy, pandas, matplotlib). Load a dataset suitable for supervised learning. Split the dataset into training and testing sets

```
#Aryan jhamnani
#229309143

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_california_housing

# Load the California Housing dataset
california_data = fetch_california_housing()

# Convert to DataFrame
california_df = pd.DataFrame(california_data.data, columns=california_data.feature_names)

# Add the target variable (house value)
california_df['TARGET'] = california_data.target

# Display basic information about the dataset
print("Dataset Information:")
print(california_df.info())

# Display the first few rows
print("First few rows:")
print(california_df.head())

# Display basic statistics
print("Basic statistics:")
print(california_df.describe())
```



Dataset Information:

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 20640 entries, 0 to 20639
```

```
Data columns (total 9 columns):
```

#	Column	Non-Null Count	Dtype
0	MedInc	20640 non-null	float64
1	HouseAge	20640 non-null	float64
2	AveRooms	20640 non-null	float64
3	AveBedrms	20640 non-null	float64
4	Population	20640 non-null	float64
5	AveOccup	20640 non-null	float64

```

6 Latitude    20640 non-null float64
7 Longitude   20640 non-null float64
8 TARGET      20640 non-null float64

```

```
dtypes: float64(9)
```

```
memory usage: 1.4 MB
```

```
None
```

```
First few rows:
```

```

MedInc HouseAge AveRooms AveBedrms Population AveOccup Latitude \
0 8.3252    41.0 6.984127 1.023810    322.0 2.555556    37.88
1 8.3014    21.0 6.238137 0.971880    2401.0 2.109842    37.86
2 7.2574    52.0 8.288136 1.073446    496.0 2.802260    37.85
3 5.6431    52.0 5.817352 1.073059    558.0 2.547945    37.85
4 3.8462    52.0 6.281853 1.081081    565.0 2.181467    37.85

```

```

Longitude TARGET
0 -122.23 4.526
1 -122.22 3.585
2 -122.24 3.521
3 -122.25 3.413
4 -122.25 3.422

```

```
Basic statistics:
```

```

MedInc HouseAge AveRooms AveBedrms Population \
count 20640.000000 20640.000000 20640.000000 20640.000000 20640.000000
mean 3.870671 28.639486 5.429000 1.096675 1425.476744
std 1.899822 12.585558 2.474173 0.473911 1132.462122
min 0.499900 1.000000 0.846154 0.333333 3.000000
25% 2.563400 18.000000 4.440716 1.006079 787.000000
50% 3.534800 29.000000 5.229129 1.048780 1166.000000
75% 4.743250 37.000000 6.052381 1.099526 1725.000000
max 15.000100 52.000000 141.909091 34.066667 35682.000000

```

```

AveOccup Latitude Longitude TARGET
count 20640.000000 20640.000000 20640.000000 20640.000000
mean 3.070655 35.631861 -119.569704 2.068558
std 10.386050 2.135952 2.003532 1.153956
min 0.692308 32.540000 -124.350000 0.149990
25% 2.429741 33.930000 -121.800000 1.196000
50% 2.818116 34.260000 -118.490000 1.797000
75% 3.282261 37.710000 -118.010000 2.647250
max 1243.333333 41.950000 -114.310000 5.000010

```

Imports:

numpy, pandas, and matplotlib are imported for data handling and visualization.

fetch_california_housing from sklearn.datasets is used to load the California Housing dataset.

Loading the Dataset:

The fetch_california_housing() function loads the dataset.

It's converted into a pandas DataFrame where the features are named according to the dataset's feature names.

The target variable (house values) is added as a new column called 'TARGET'.

Displaying Information:

info(): Displays summary information about the dataset (e.g., number of rows, column types, non-null values).

head(): Shows the first few rows of the dataset to give a preview.

describe(): Provides basic statistics such as mean, min, max, and standard deviation for each feature.

```
#Aryan jhamnani
#229309143
# Separate features and target
X = california_df.drop('TARGET', axis=1)
y = california_df['TARGET']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Scale the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Visualize the distribution of house values
plt.figure(figsize=(10, 6))
plt.hist(y, bins=30, edgecolor='black', alpha=0.7)
plt.title('Distribution of House Values')
plt.xlabel('Median House Value ($100,000s)')
plt.ylabel('Frequency')
plt.grid(True, alpha=0.3)
plt.show()

# Visualize correlations between features using Matplotlib
plt.figure(figsize=(12, 8))
correlation_matrix = california_df.corr()

# Plot correlation matrix as a heatmap
plt.imshow(correlation_matrix, cmap='coolwarm', aspect='auto')
plt.colorbar()

# Set axis labels
plt.xticks(range(len(correlation_matrix.columns)), correlation_matrix.columns, rotation=45, ha='right')
plt.yticks(range(len(correlation_matrix.columns)), correlation_matrix.columns)

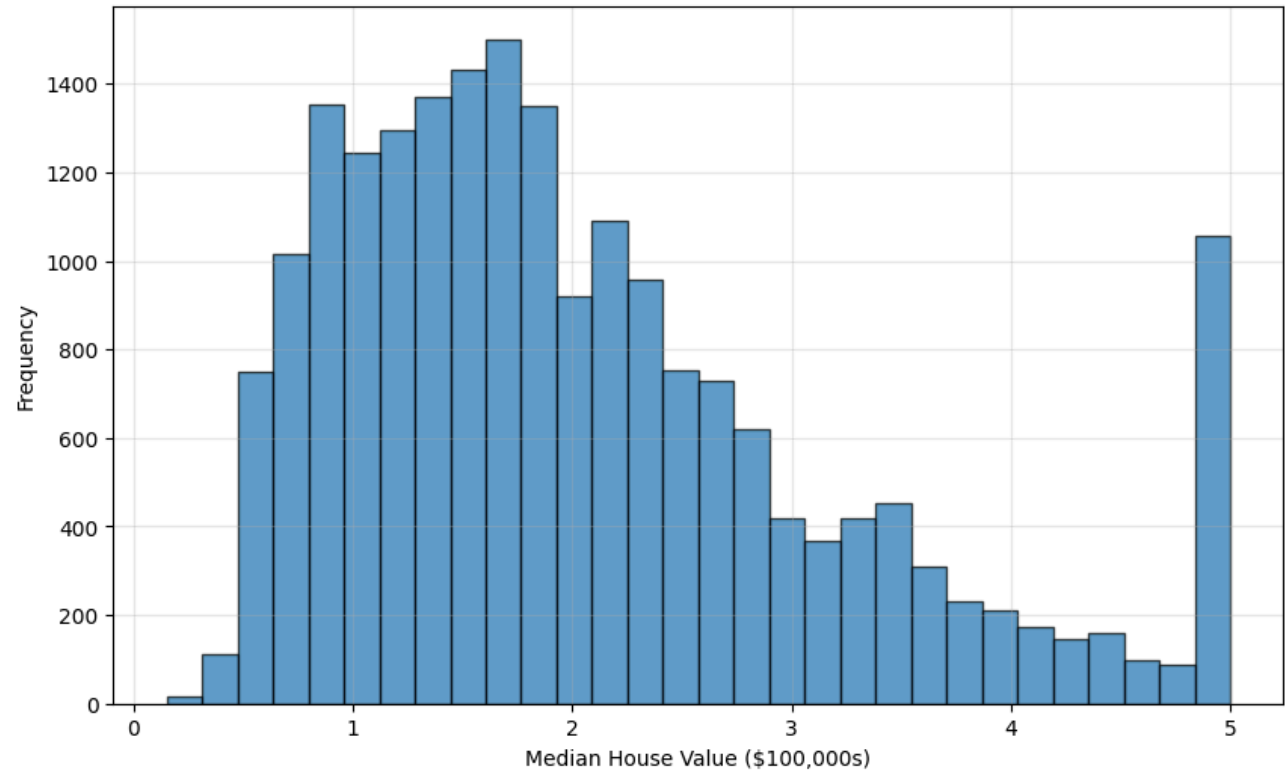
plt.title('Feature Correlation Matrix')
plt.tight_layout()
plt.show()

# Print dataset shape
```

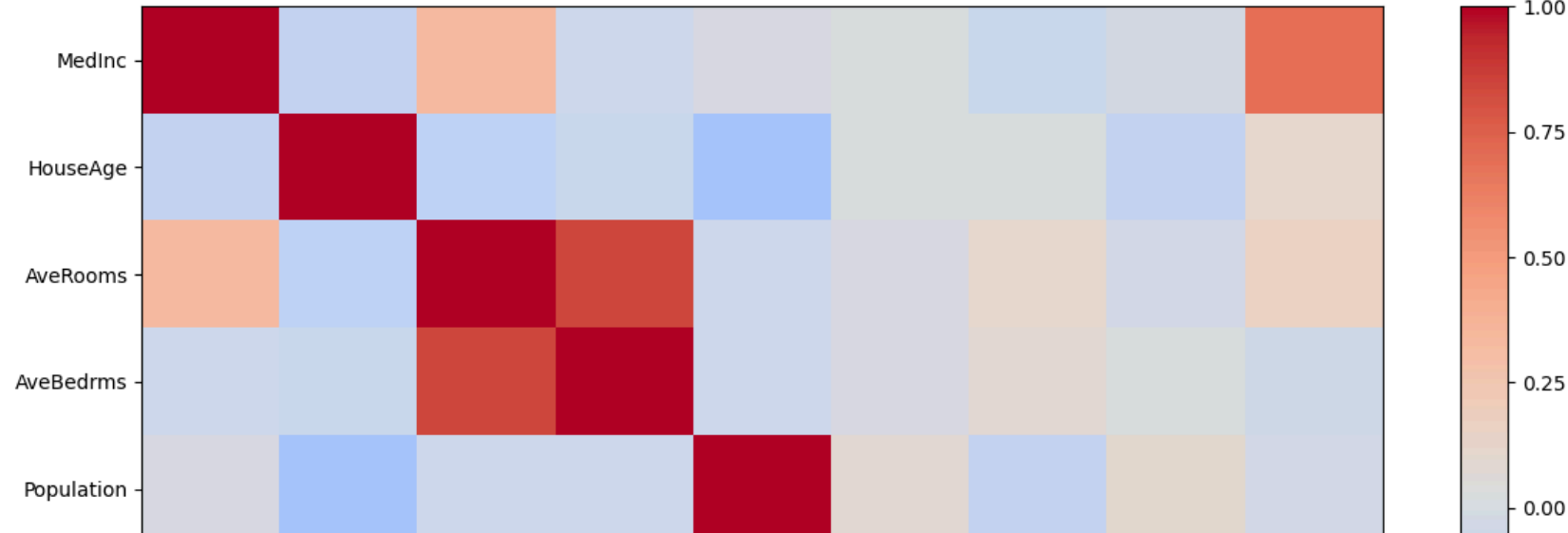
```
print("Shape of training set:", X_train.shape)  
print("Shape of testing set:", X_test.shape)
```

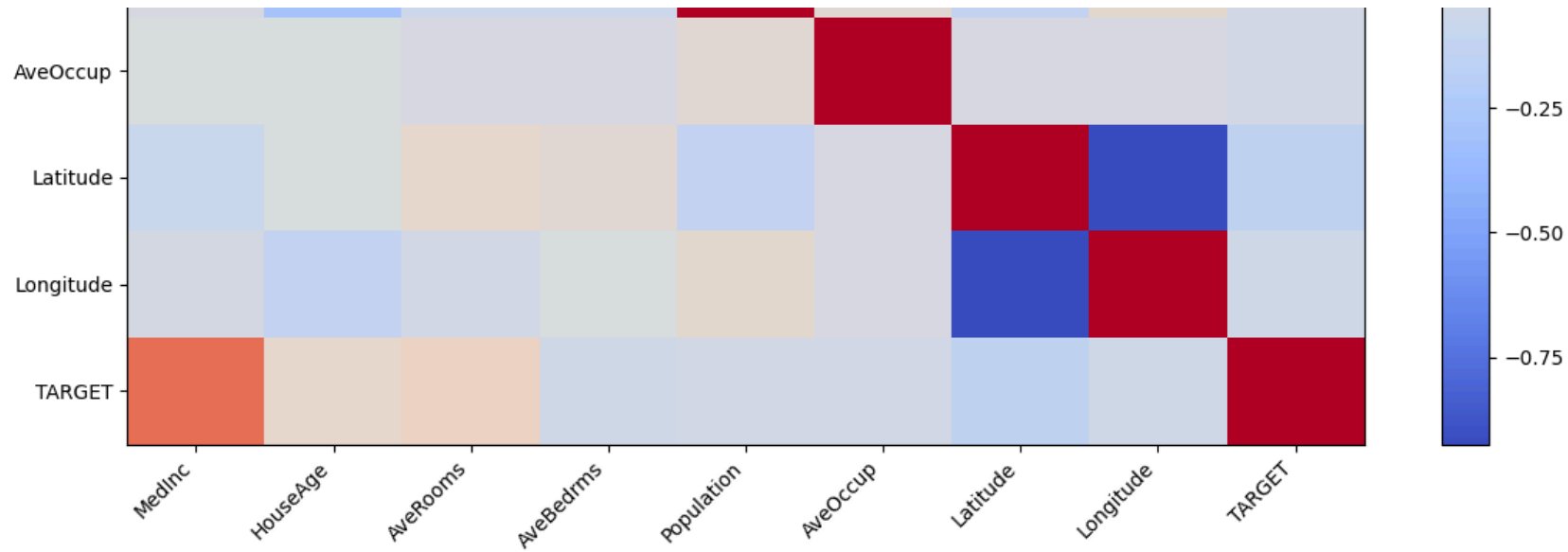


Distribution of House Values



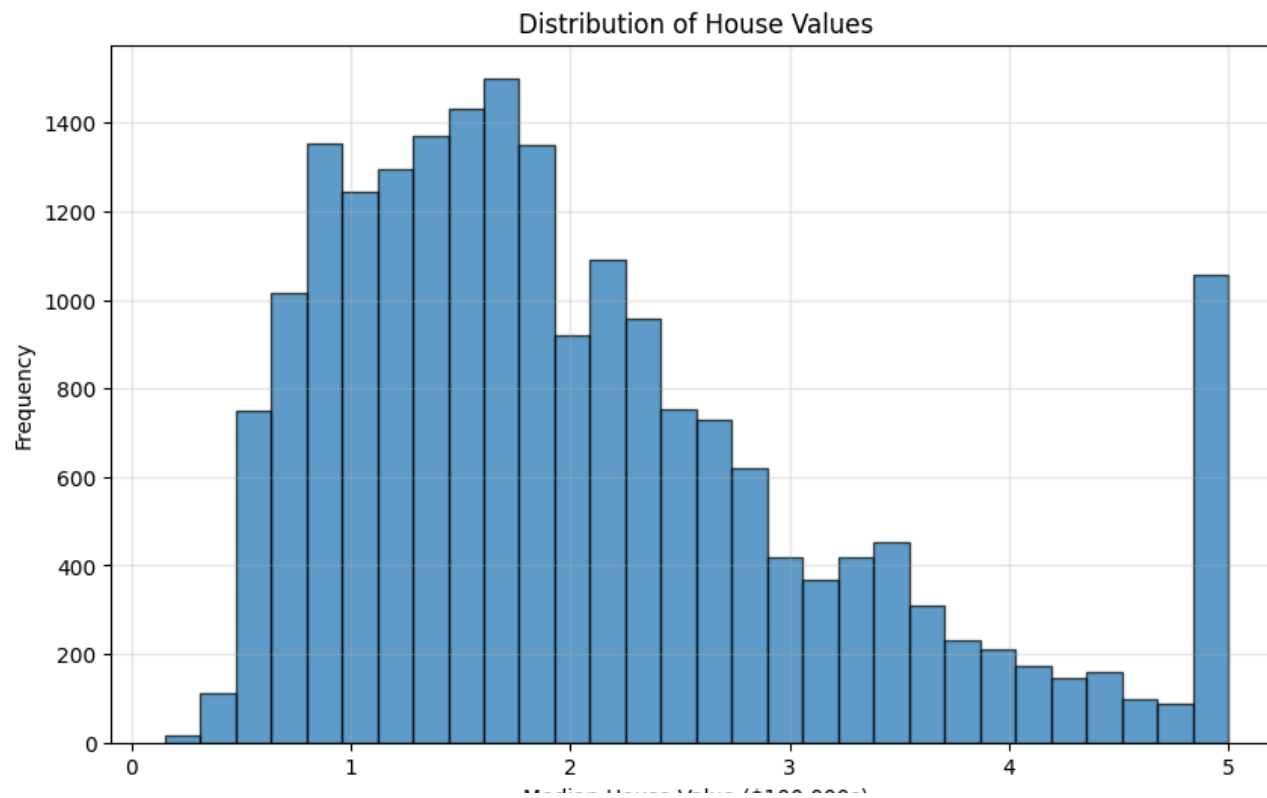
Feature Correlation Matrix



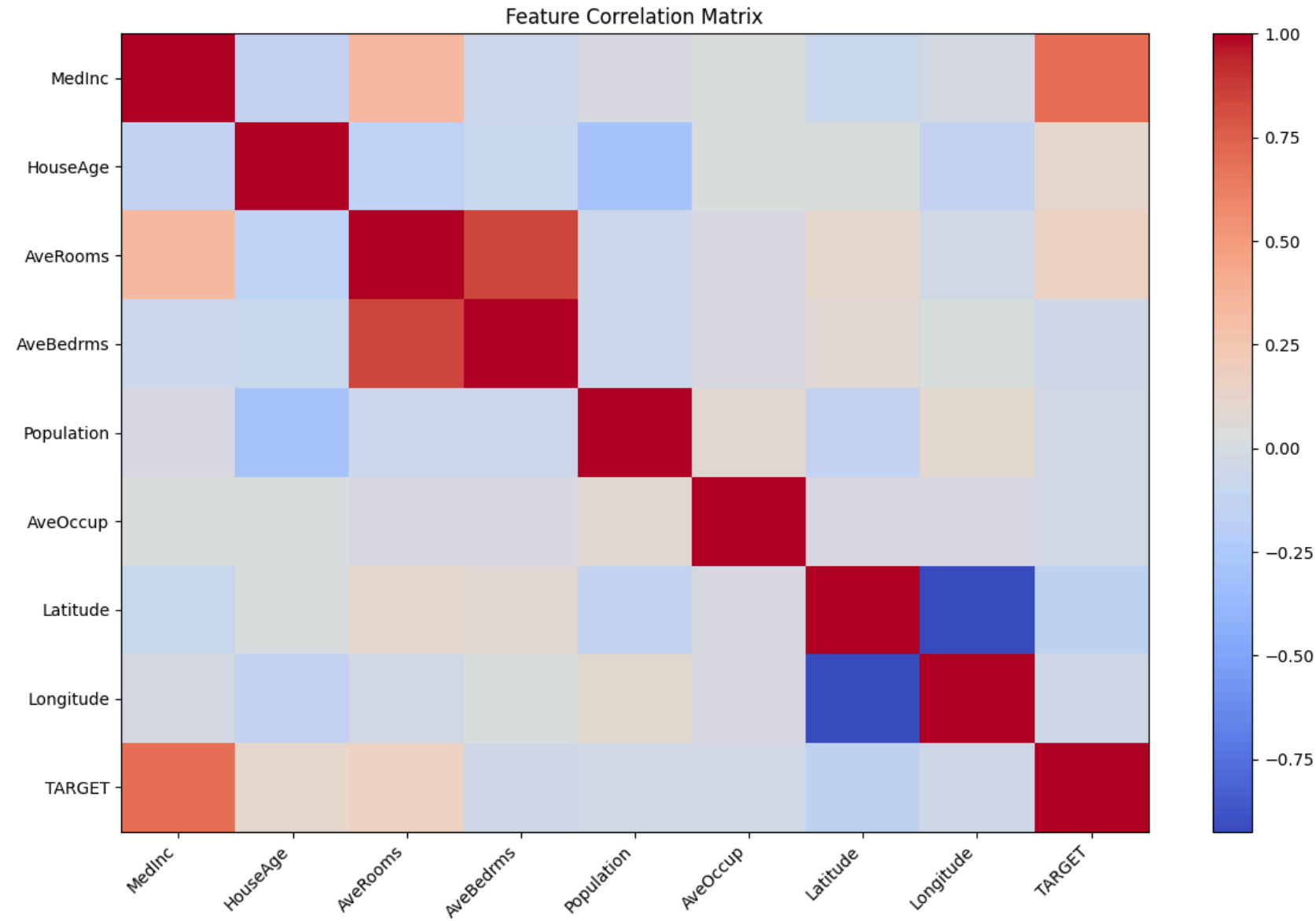


Shape of training set: (16512, 8)

Shape of testing set: (4128, 8)



Median House value (\$100,000s)



Shape of training set: (16512, 8)
Shape of testing set: (4128, 8)

Data Splitting & Scaling:

Splits the dataset into 80% training and 20% testing using `train_test_split()`.

Standardizes the features using `StandardScaler()` to improve model performance.

Visualization:

Histogram of House Values (`plt.hist()`):

Shows how house prices are distributed.

Feature Correlation Matrix (`plt.imshow()`):

Displays relationships between different features using a heatmap.

Lab 2: Implement a basic gradient descent algorithm for a simple function. Visualize the convergence of the algorithm over iterations.

```
#Aryan jhamnani
#229309143
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Load California Housing dataset
california_data = fetch_california_housing()
X = pd.DataFrame(california_data.data, columns=california_data.feature_names)
y = california_data.target.reshape(-1, 1) # Reshape y for matrix operations

# Add bias term (intercept)
X['bias'] = 1

# Convert to NumPy arrays
X = X.values

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Scale the features (excluding bias)
scaler = StandardScaler()
X_train[:, :-1] = scaler.fit_transform(X_train[:, :-1])
X_test[:, :-1] = scaler.transform(X_test[:, :-1])

# Initialize parameters (theta)
theta = np.random.randn(X_train.shape[1], 1)
```

```

# Define the Mean Squared Error cost function
def compute_cost(X, y, theta):
    m = len(y)
    predictions = X @ theta
    cost = (1 / (2 * m)) * np.sum((predictions - y) ** 2)
    return cost

# Define the gradient function
def compute_gradient(X, y, theta):
    m = len(y)
    gradients = (1 / m) * X.T @ (X @ theta - y)
    return gradients

# Implement gradient descent
def gradient_descent(X, y, theta, learning_rate, n_iterations):
    cost_history = []
    theta_history = [theta.copy()]

    for i in range(n_iterations):
        gradients = compute_gradient(X, y, theta)
        theta -= learning_rate * gradients # Update weights
        cost_history.append(compute_cost(X, y, theta))
        theta_history.append(theta.copy())

    return theta, cost_history, theta_history

# Set hyperparameters
learning_rate = 0.1
n_iterations = 100

# Run gradient descent
theta_final, cost_history, theta_history = gradient_descent(X_train, y_train, theta, learning_rate, n_iterations)

# Visualization
plt.figure(figsize=(12, 6))

# Plot 1: Cost function convergence
plt.subplot(1, 2, 1)
plt.plot(range(n_iterations), cost_history, 'r-o')
plt.title('Gradient Descent Convergence')
plt.xlabel('Iteration')
plt.ylabel('Cost (MSE)')
plt.grid(True)

# Plot 2: Predicted vs Actual Values (Test Set)
y_pred = X_test @ theta_final
plt.subplot(1, 2, 2)
plt.scatter(y_test, y_pred, alpha=0.5)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r') # Ideal line
plt.title('Actual vs Predicted House Prices')

```

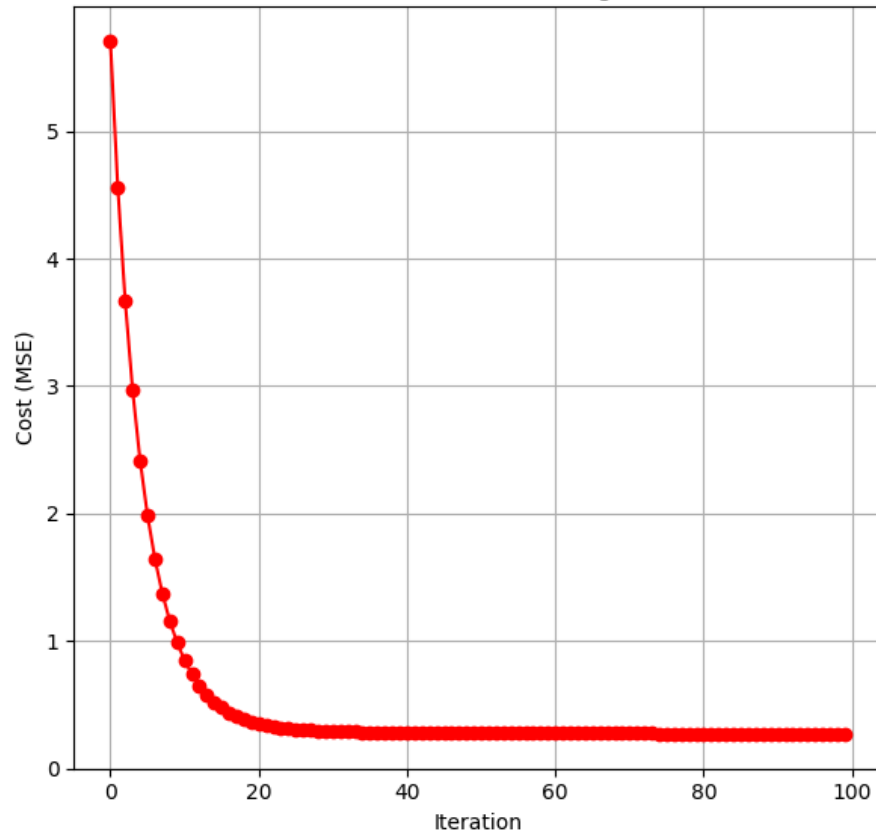
```
plt.xlabel('Actual Prices')
plt.ylabel('Predicted Prices')
plt.grid(True)

plt.tight_layout()
plt.show()

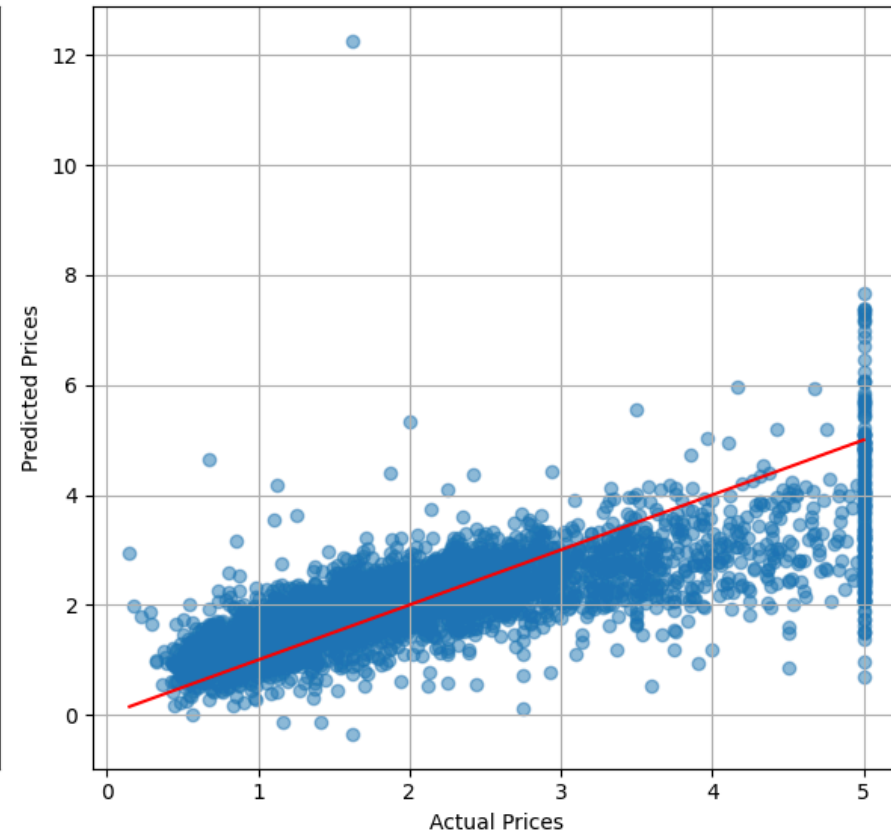
# Print final results
print(f"Final parameters (theta):\n{theta_final}")
print(f"Final cost: {cost_history[-1]:.6f}")
```



Gradient Descent Convergence



Actual vs Predicted House Prices

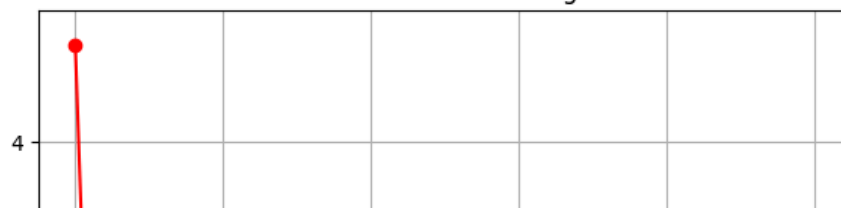


Final parameters (theta):

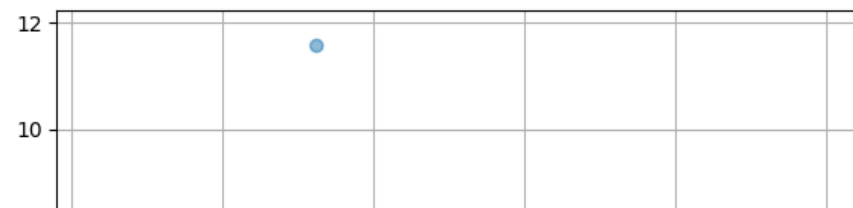
```
[[ 0.97800301]
 [ 0.16842273]
 [-0.47916241]
 [ 0.470332 ]
 [ 0.01252865]
 [-0.04826301]
 [-0.44839652]
 [-0.43391526]
 [ 2.0719176 ]]
```

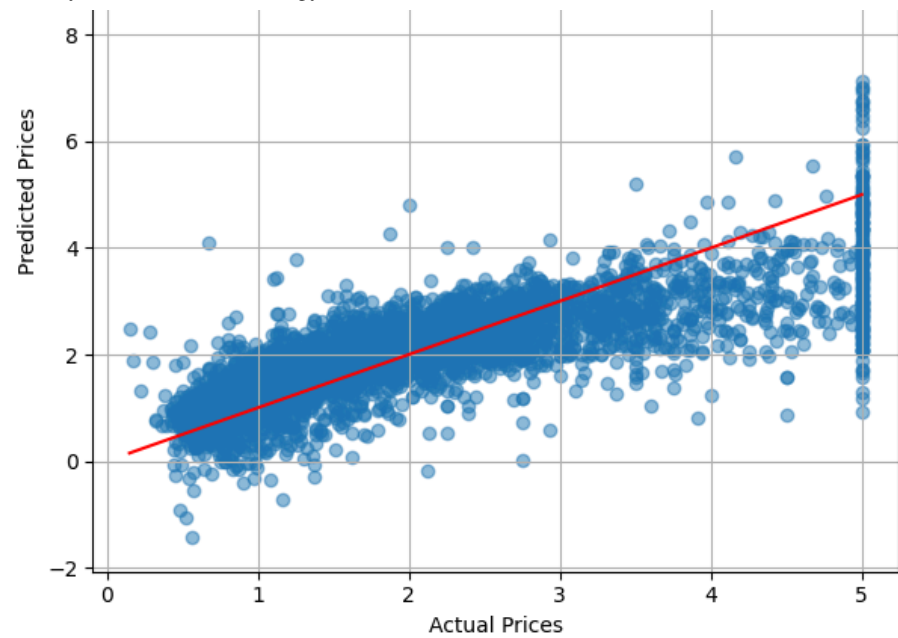
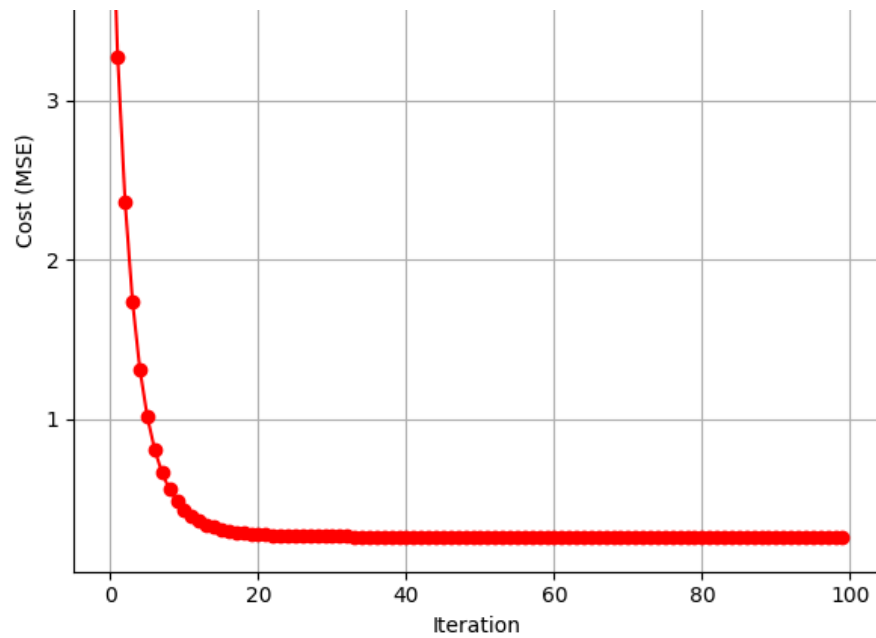
Final cost: 0.270444

Gradient Descent Convergence



Actual vs Predicted House Prices





Final parameters (theta):

```
[[ 0.81463888]  
 [ 0.10366165]  
 [-0.24248972]  
 [ 0.30691015]  
 [-0.00849017]  
 [-0.0380904 ]  
 [-1.07252152]  
 [-1.04175425]  
 [ 2.07191857]]
```

Final cost: 0.260692

Lab 3: Implement a decision tree classifier using scikit-learn. Tune the hyperparameters of the decision tree. Visualize the constructed decision tree.

```
#Aryan jhamnani
#229309143
# Import necessary libraries for Decision Tree
from sklearn.tree import DecisionTreeRegressor, export_graphviz
from sklearn.model_selection import GridSearchCV
import graphviz

# Define the Decision Tree model
dt = DecisionTreeRegressor(random_state=42)

# Define hyperparameter grid
param_grid = {
    "max_depth": [3, 5, 10, None],
    "min_samples_split": [2, 5, 10],
    "min_samples_leaf": [1, 2, 4]
}

# Perform Grid Search with Cross-Validation
grid_search = GridSearchCV(dt, param_grid, cv=5, scoring='neg_mean_squared_error')
grid_search.fit(X_train, y_train.ravel()) # Flatten y for scikit-learn compatibility

# Get the best model
best_dt = grid_search.best_estimator_

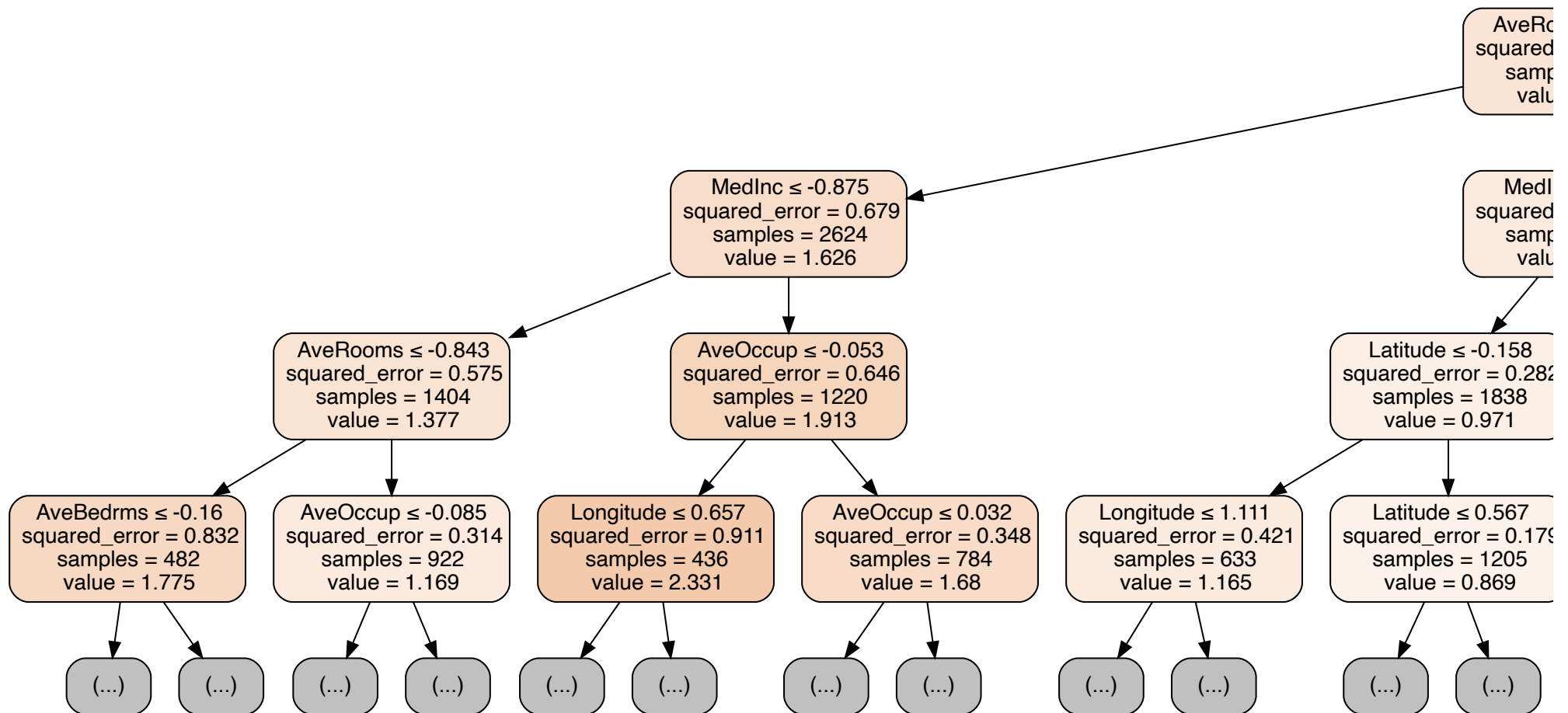
# Print best parameters
print("Best Parameters:", grid_search.best_params_)

# Use the original feature names
feature_names = california_data.feature_names + ['bias']

# Visualize the best Decision Tree with smaller size
dot_data = export_graphviz(best_dt, out_file=None,
                           feature_names=feature_names,
                           filled=True, rounded=True,
                           special_characters=True,
                           max_depth=5) # Limit depth for better readability

# Convert to graph and display
graph = graphviz.Source(dot_data)
graph.render("california_housing_tree") # Saves as a .png file
graph # Displays the decision tree in the notebook
```

➡ Best Parameters: {'max_depth': 10, 'min_samples_leaf': 4, 'min_samples_split': 10}



Lab4: Implement a Support Vector Machine using scikit-learn. Experiment with different kernel functions. Visualize the decision boundaries.

```
#Aryan jhamnani
#229309143
# Import necessary libraries
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error
import numpy as np
import matplotlib.pyplot as plt

# Define different kernels to experiment with
kernels = ['linear', 'poly', 'rbf']

# Store results
results = {}

# Train and evaluate SVR models for each kernel
for kernel in kernels:
    svr = SVR(kernel=kernel)
    svr.fit(X_train_scaled, y_train.ravel()) # Flatten y for SVR compatibility

    # Make predictions
    y_pred = svr.predict(X_test_scaled)

    # Compute Mean Squared Error (MSE)
    mse = mean_squared_error(y_test, y_pred)

    # Store results
    results[kernel] = (svr, mse, y_pred)

    # Print MSE
    print(f"Kernel: {kernel}, MSE: {mse:.4f}")

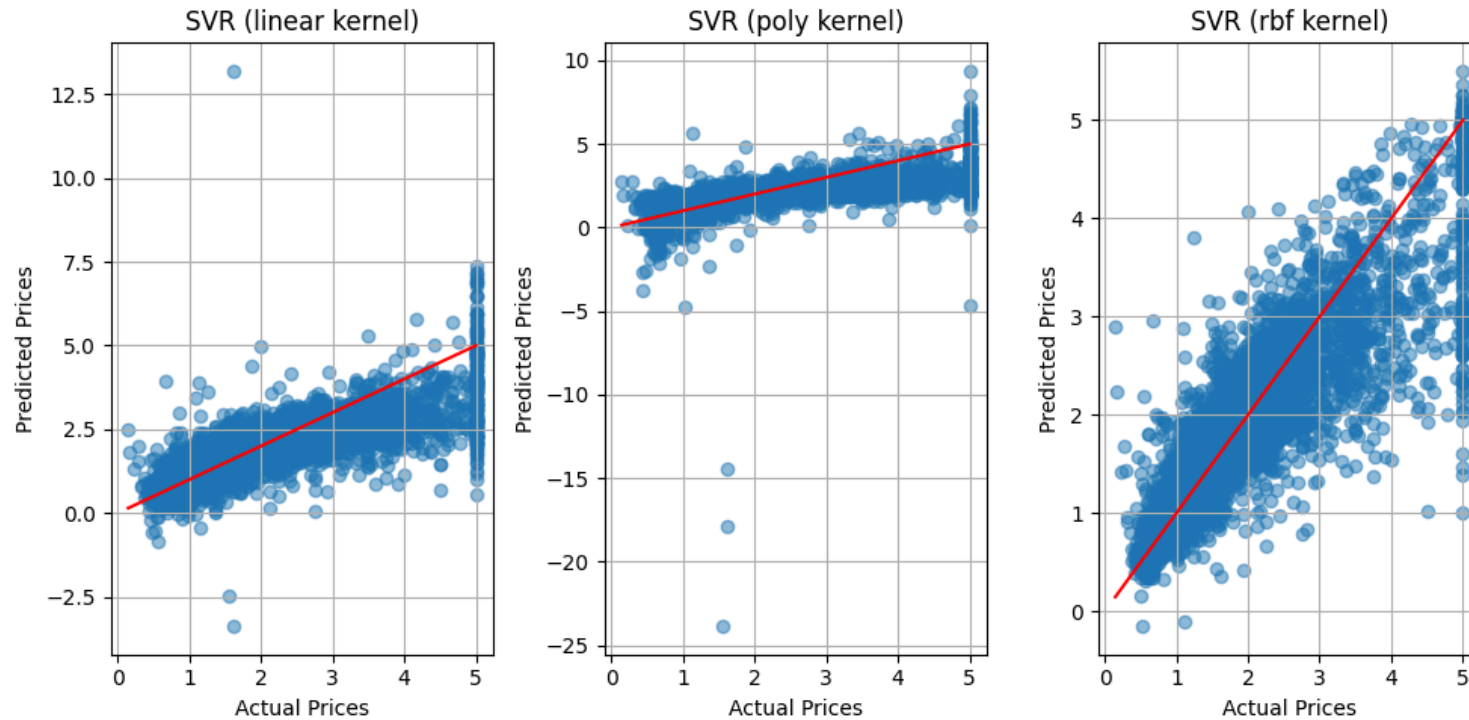
# Visualization of Predictions vs Actual Prices
plt.figure(figsize=(10, 5))

for i, (kernel, (model, mse, y_pred)) in enumerate(results.items()):
    plt.subplot(1, 3, i + 1)
    plt.scatter(y_test, y_pred, alpha=0.5)
    plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r') # Ideal line
    plt.title(f'SVR ({kernel} kernel)')
    plt.xlabel('Actual Prices')
    plt.ylabel('Predicted Prices')
    plt.grid(True)

plt.tight_layout()
```

```
plt.show()
```

```
Kernel: linear, MSE: 0.5793
Kernel: poly, MSE: 1.0031
Kernel: rbf, MSE: 0.3570
```



Lab 5: Experiment with different regularization parameters. Implement Logistic Regression for classification tasks.

```
#Aryan jhamnani
#229309143
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, KBinsDiscretizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report

# Load California Housing Dataset
california = fetch_california_housing()
X = pd.DataFrame(california.data, columns=california.feature_names)
```

```

y = california.target # Continuous target variable (Median House Value)

# Convert continuous target into categories (Low, Medium, High price categories)
binner = KBinsDiscretizer(n_bins=3, encode='ordinal', strategy='quantile')
y_binned = binner.fit_transform(y.reshape(-1, 1)).astype(int).flatten() # Convert to integer labels (0,1,2)

# Split dataset (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y_binned, test_size=0.2, random_state=42, stratify=y_binned)

# Standardize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Define regularization parameters
C_values = [0.001, 0.01, 0.1, 1, 10, 100]
regularizations = ['l1', 'l2']

# Store results
results = []

# Train and evaluate models with different regularization parameters
for reg in regularizations:
    for C in C_values:
        try:
            # Train Logistic Regression model
            lr = LogisticRegression(C=C, penalty=reg, solver='saga', max_iter=1000, random_state=42)
            lr.fit(X_train_scaled, y_train)

            # Make predictions
            y_pred = lr.predict(X_test_scaled)

            # Calculate accuracy
            accuracy = accuracy_score(y_test, y_pred)

            # Store results
            results.append({'regularization': reg, 'C': C, 'accuracy': accuracy})
        except Exception as e:
            print(f"Skipping {reg} with C={C} due to error: {e}")

# Convert results to DataFrame
results_df = pd.DataFrame(results)

# Plot results
plt.figure(figsize=(10, 6))
for reg in regularizations:
    reg_results = results_df[results_df['regularization'] == reg]
    plt.semilogx(reg_results['C'], reg_results['accuracy'], marker='o', label=f'{reg} regularization')

plt.xlabel('Regularization parameter (C)')

```

```
plt.ylabel('Accuracy')
plt.title('Logistic Regression Performance with Different Regularization Parameters')
plt.legend()
plt.grid(True)
plt.show()

# Print detailed results for the best model
best_result = max(results, key=lambda x: x['accuracy'])
print(f"\nBest model parameters:")
print(f"Regularization: {best_result['regularization']}")
print(f"C: {best_result['C']}")
print(f"Accuracy: {best_result['accuracy']:.4f}")

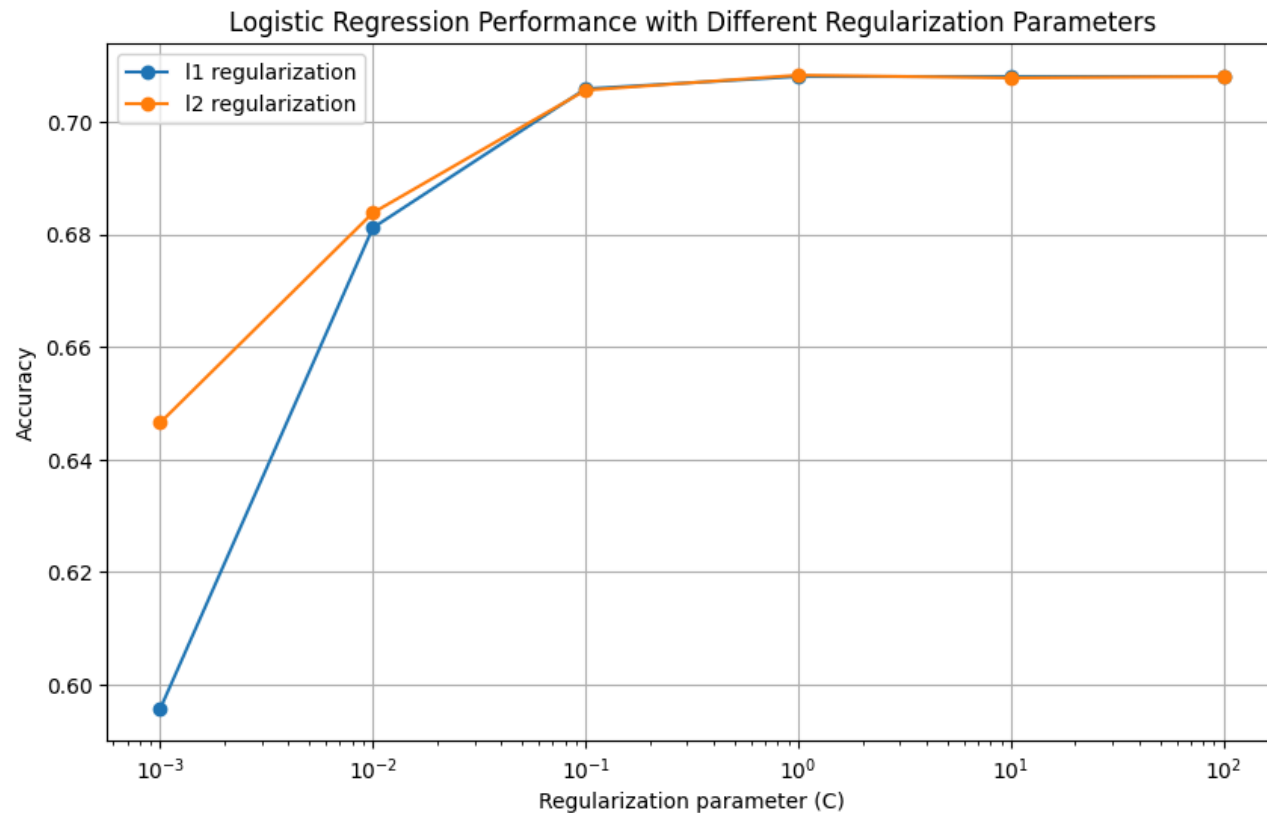
# Train the best model and show detailed classification report
best_lr = LogisticRegression(C=best_result['C'], penalty=best_result['regularization'], solver='saga', max_iter=1000, random_state=42)
best_lr.fit(X_train_scaled, y_train)
y_pred = best_lr.predict(X_test_scaled)

print("\nClassification Report for Best Model:")
print(classification_report(y_test, y_pred))

# Feature Importance Plot
feature_importance = pd.DataFrame({'feature': X.columns, 'importance': np.abs(best_lr.coef_).mean(axis=0)})
feature_importance = feature_importance.sort_values('importance', ascending=False)

plt.figure(figsize=(10, 6))
plt.bar(feature_importance['feature'], feature_importance['importance'])
plt.xticks(rotation=45, ha='right')
plt.title('Feature Importance in Logistic Regression')
plt.tight_layout()
plt.show()
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_sag.py:348: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_sag.py:348: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_sag.py:348: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_sag.py:348: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_sag.py:348: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_sag.py:348: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_sag.py:348: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
warnings.warn(
```



Best model parameters:

Regularization: l2

C: 1

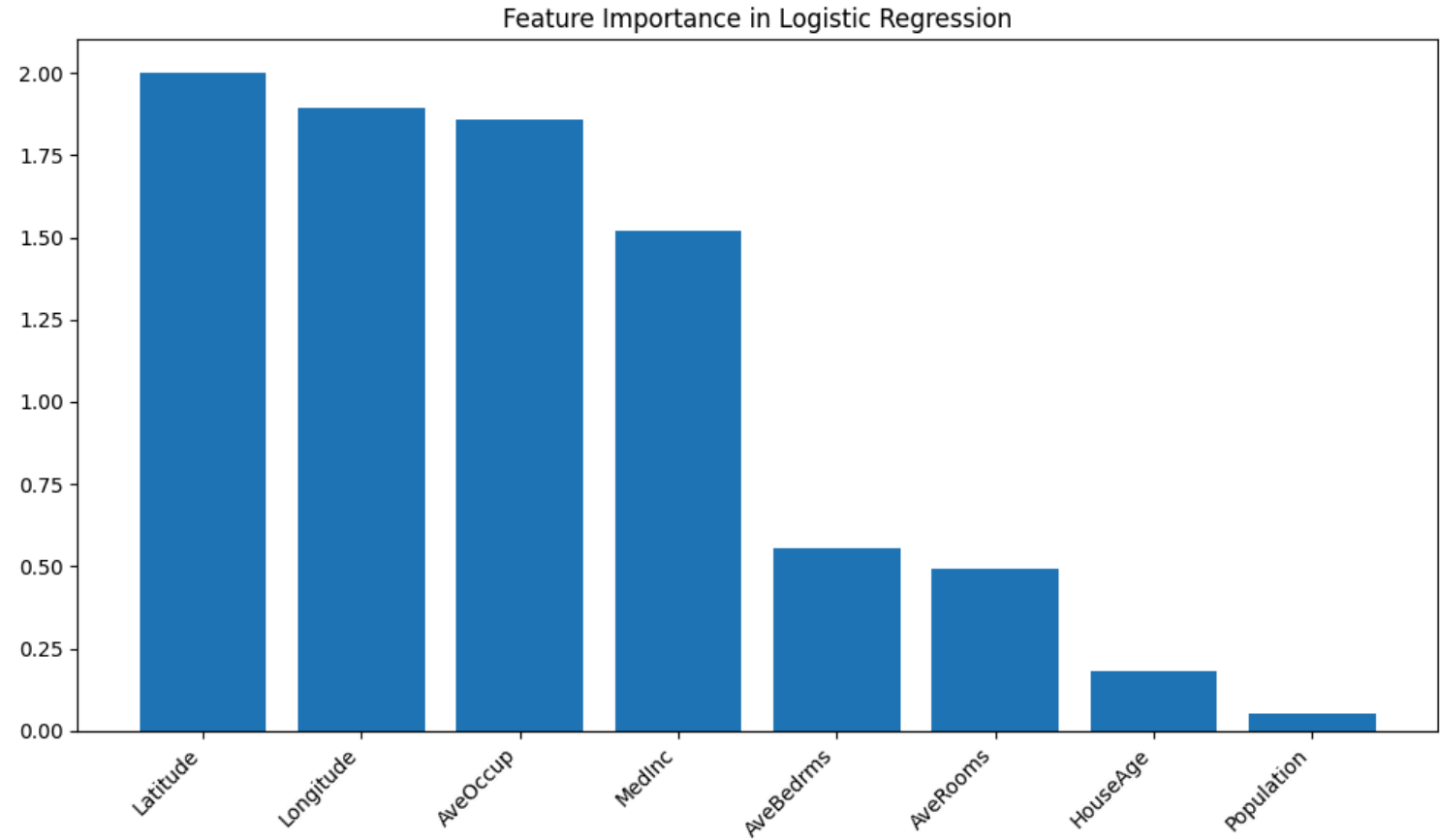
Accuracy: 0.7083

Classification Report for Best Model:

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

	0	0.78	0.75	0.76	1375
	1	0.59	0.61	0.60	1377
	2	0.77	0.77	0.77	1376
accuracy				0.71	4128
macro avg		0.71	0.71	0.71	4128
weighted avg		0.71	0.71	0.71	4128

/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_sag.py:348: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
warnings.warn(



Lab 6: Implement a basic ensemble method (e.g., Random Forest) using scikit-learn. Implement Adaptive Boosting (AdaBoost) and Extreme Gradient Boosting (XGBoost). Compare the performance of individual models with the ensemble.

```
#Aryan jhamnani
#229309143
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from xgboost import XGBClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report

# Define base classifier (Decision Tree)
dt = DecisionTreeClassifier(random_state=42)

# Define ensemble models
rf = RandomForestClassifier(n_estimators=100, random_state=42)
ada = AdaBoostClassifier(n_estimators=100, random_state=42)
xgb = XGBClassifier(n_estimators=100, use_label_encoder=False, eval_metric='mlogloss', random_state=42)

# Train and evaluate each model
models = {'Decision Tree': dt, 'Random Forest': rf, 'AdaBoost': ada, 'XGBoost': xgb}
results = {}

for name, model in models.items():
    model.fit(X_train_scaled, y_train) # Train
    y_pred = model.predict(X_test_scaled) # Predict
    accuracy = accuracy_score(y_test, y_pred) # Accuracy
    results[name] = accuracy
    print(f"\n{name} Model:")
    print(f"Accuracy: {accuracy:.4f}")
    print(classification_report(y_test, y_pred))

# Plot model performance comparison
plt.figure(figsize=(8, 5))
plt.bar(results.keys(), results.values(), color=['blue', 'green', 'red', 'purple'])
plt.ylabel("Accuracy")
plt.title("Comparison of Individual vs. Ensemble Models")
plt.ylim(0.5, 1)
plt.show()
```



Decision Tree Model:

Accuracy: 0.7321

	precision	recall	f1-score	support
0	0.80	0.77	0.78	1375
1	0.63	0.66	0.64	1377
2	0.78	0.76	0.77	1376
accuracy			0.73	4128
macro avg	0.73	0.73	0.73	4128
weighted avg	0.73	0.73	0.73	4128

Random Forest Model:

Accuracy: 0.8018

	precision	recall	f1-score	support
0	0.84	0.87	0.85	1375
1	0.72	0.72	0.72	1377
2	0.85	0.82	0.83	1376
accuracy			0.80	4128
macro avg	0.80	0.80	0.80	4128
weighted avg	0.80	0.80	0.80	4128

AdaBoost Model:

Accuracy: 0.7175

	precision	recall	f1-score	support
0	0.80	0.72	0.76	1375
1	0.58	0.69	0.63	1377
2	0.82	0.74	0.77	1376
accuracy			0.72	4128
macro avg	0.73	0.72	0.72	4128
weighted avg	0.73	0.72	0.72	4128

/usr/local/lib/python3.11/dist-packages/xgboost/core.py:158: UserWarning: [19:38:47] WARNING: /workspace/src/learner.cc:740: Parameters: { "use_label_encoder" } are not used.

warnings.warn(smsg, UserWarning)

XGBoost Model:

Accuracy: 0.8297

	precision	recall	f1-score	support
0	0.87	0.87	0.87	1375
1	0.75	0.77	0.76	1377
2	0.88	0.84	0.86	1376
accuracy			0.83	4128
macro avg	0.83	0.83	0.83	4128
weighted avg	0.83	0.83	0.83	4128

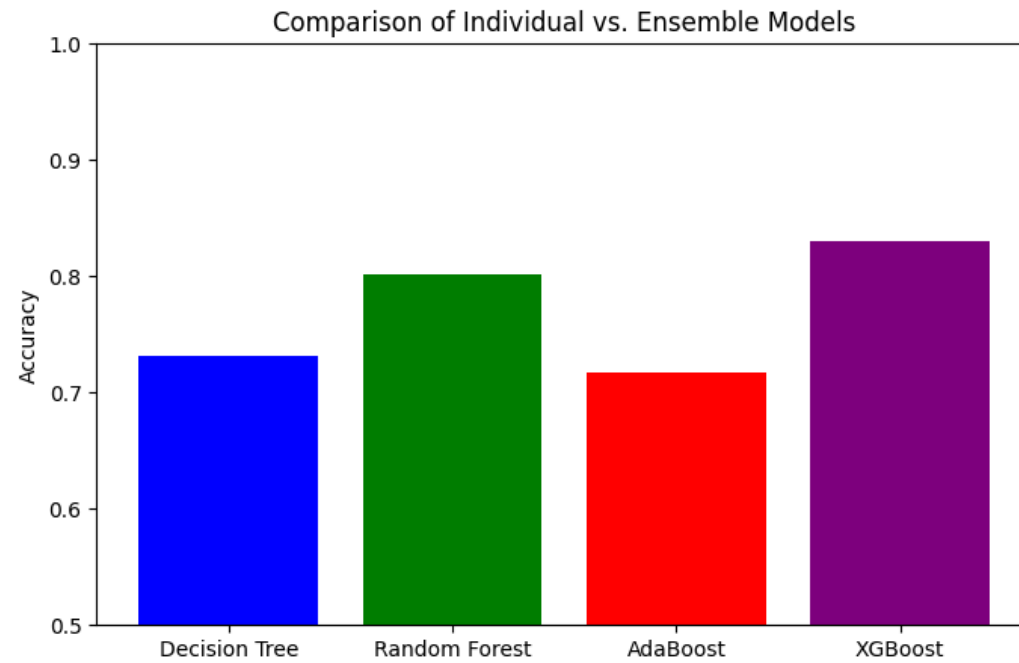
weighted avg

0.85

0.85

0.85

4120



Lab 7: Implement k-Means clustering and Hierarchical clustering. Implement a density-based clustering algorithm (e.g., DBSCAN). Visualize the clustering results.

```
#Aryan jhamnani
#229309143
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans, AgglomerativeClustering, DBSCAN
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from scipy.cluster.hierarchy import dendrogram, linkage

# Load California Housing Data
from sklearn.datasets import fetch_california_housing
data = fetch_california_housing()
df = pd.DataFrame(data.data, columns=data.feature_names)

# Standardize the features
scaler = StandardScaler()
df_scaled = scaler.fit_transform(df)

# Reduce dimensions to 2D for visualization using PCA
pca = PCA(n_components=2)
df_pca = pca.fit_transform(df_scaled)

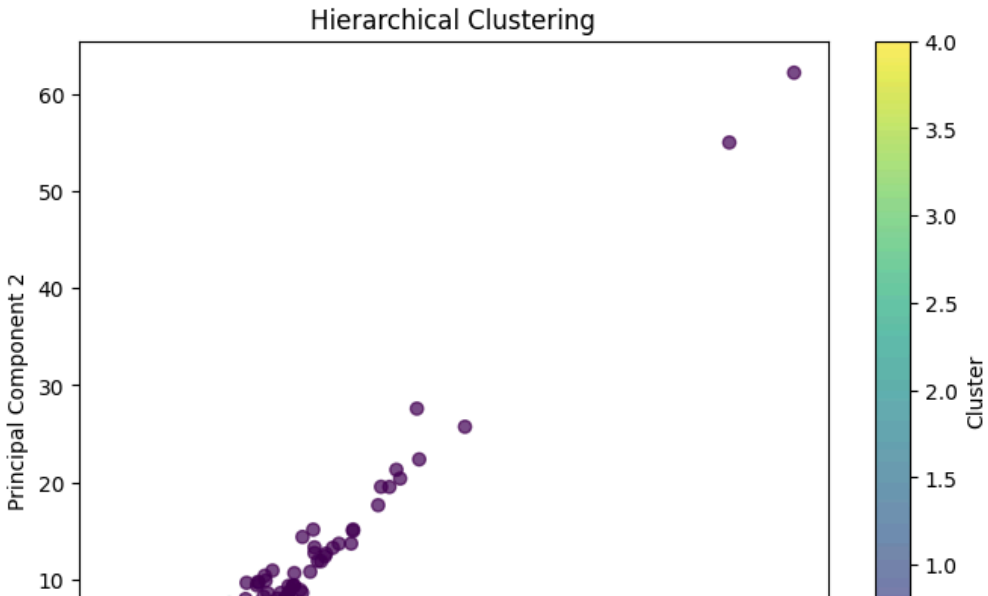
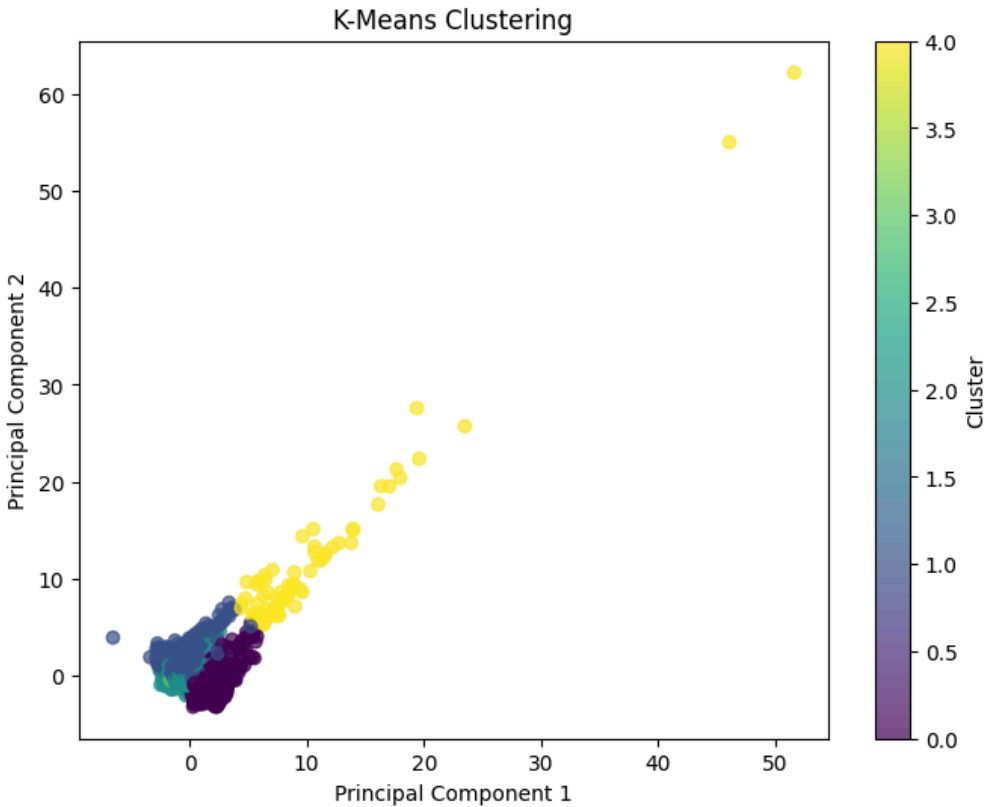
# Function to visualize clusters
def plot_clusters(labels, title):
    plt.figure(figsize=(8,6))
    plt.scatter(df_pca[:, 0], df_pca[:, 1], c=labels, cmap='viridis', alpha=0.7)
    plt.xlabel("Principal Component 1")
    plt.ylabel("Principal Component 2")
    plt.title(title)
    plt.colorbar(label="Cluster")
    plt.show()

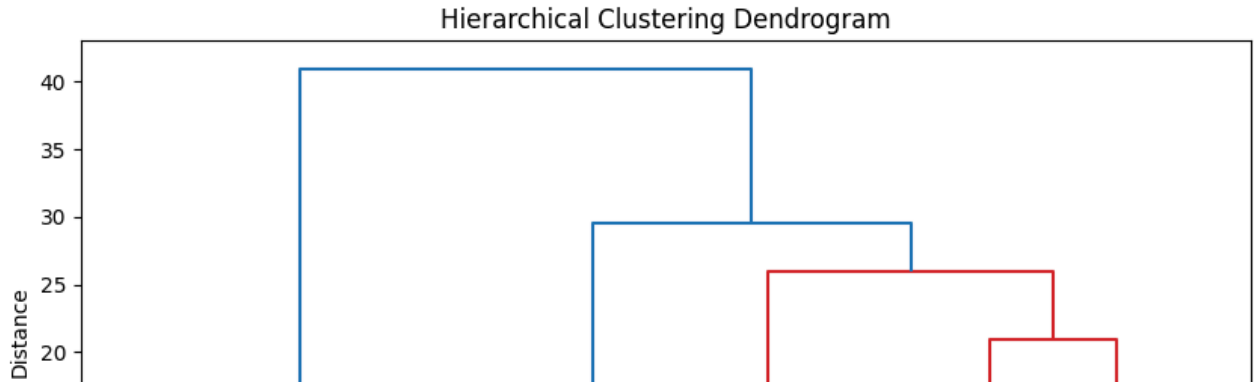
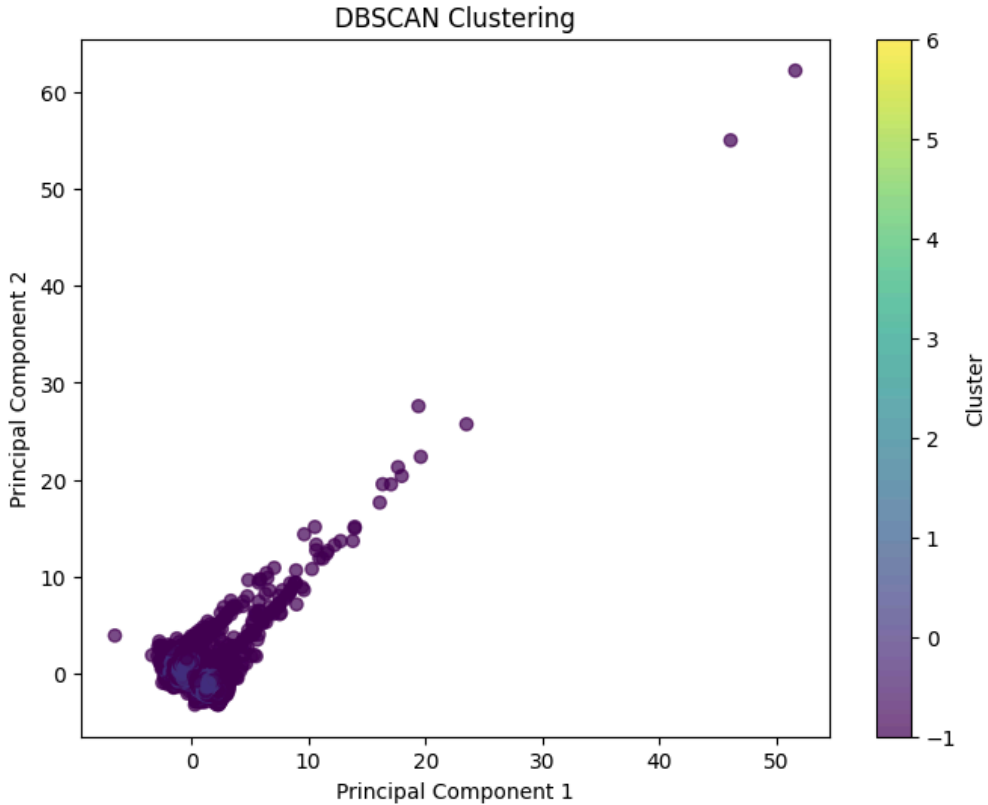
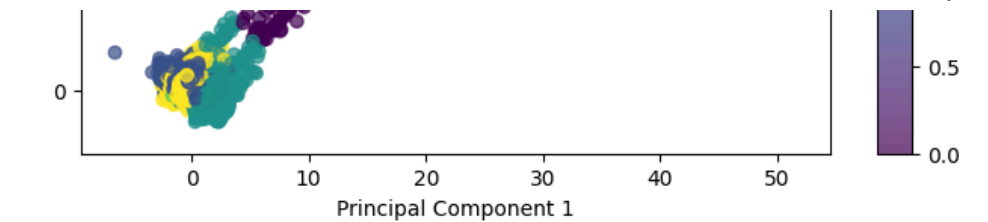
# K-Means Clustering
kmeans = KMeans(n_clusters=5, random_state=42, n_init=10)
kmeans_labels = kmeans.fit_predict(df_scaled)
plot_clusters(kmeans_labels, "K-Means Clustering")

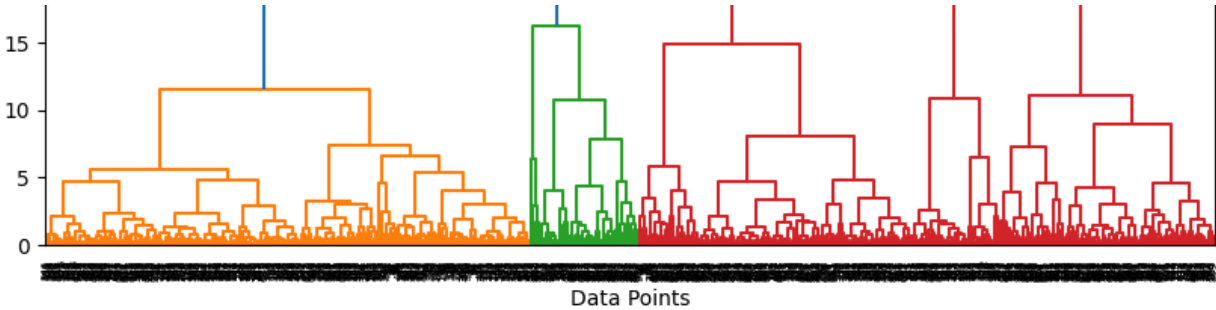
# Hierarchical Clustering
hierarchical = AgglomerativeClustering(n_clusters=5)
hierarchical_labels = hierarchical.fit_predict(df_scaled)
plot_clusters(hierarchical_labels, "Hierarchical Clustering")
```

```
# DBSCAN Clustering
dbscan = DBSCAN(eps=0.5, min_samples=10)
dbscan_labels = dbscan.fit_predict(df_scaled)
plot_clusters(dbscan_labels, "DBSCAN Clustering")

# Hierarchical Dendrogram
plt.figure(figsize=(10, 5))
Z = linkage(df_scaled[:1000], method='ward') # Limit to 1000 samples for efficiency
dendrogram(Z)
plt.title("Hierarchical Clustering Dendrogram")
plt.xlabel("Data Points")
plt.ylabel("Distance")
plt.show()
```







Lab 8:Implement k-N). Visualize KNN for a simple classification task and compare its performance with a decision tree model. Analyze the effect of varying K on accuracy.

```
#Aryan jhamnani
#229309143
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, KBinsDiscretizer
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Step 1: Load California Housing dataset
housing = fetch_california_housing()
X = housing.data
y = housing.target # Continuous target (median house value)

# Step 2: Convert target variable into categories (classification task)
discretizer = KBinsDiscretizer(n_bins=3, encode='ordinal', strategy='quantile')
y_binned = discretizer.fit_transform(y.reshape(-1, 1)).astype(int).flatten()

# Step 3: Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y_binned, test_size=0.2, random_state=42, stratify=y_binned)

# Step 4: Scale features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Step 5: Train KNN with different K values and Decision Tree
k_values = [1, 5, 10]
knn_accuracies = []

for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train_scaled, y_train)
    y_pred_knn = knn.predict(X_test_scaled)
    knn_accuracies.append(accuracy_score(y_test, y_pred_knn))

# Train Decision Tree Classifier
dtree = DecisionTreeClassifier(max_depth=5, random_state=42)
dtree.fit(X_train_scaled, y_train)
y_pred_tree = dtree.predict(X_test_scaled)
dtree_accuracy = accuracy_score(y_test, y_pred_tree)
```