HACETTEPE UNIVERSITY

DEPARTMENT OF ELECTRICAL & ELECTRONICS ENGINEERING

ELE489 – Fundamentals of Machine Learning

Homework 1-Report

Hamza Emir Tarcan

2200357073

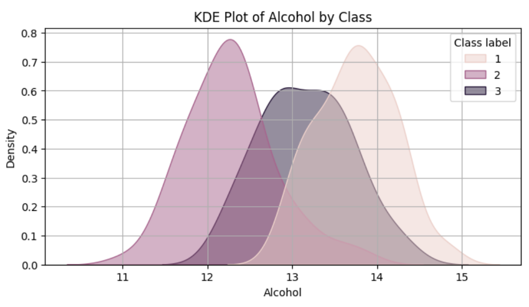
1. Introduction

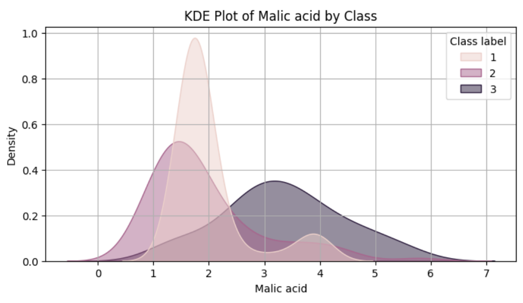
In this homework assignment, the k-Nearest Neighbors (k-NN) algorithm was implemented from scratch using Python and applied to the Wine dataset from the UCI Machine Learning Repository. The goal of the project was to explore the behavior of the k-NN algorithm under different parameter settings, specifically varying values of k and distance metrics, and to evaluate its performance on a real-world classification task.

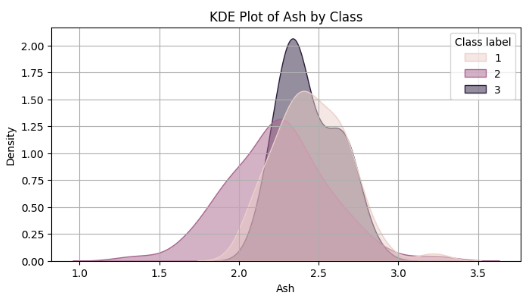
To begin with, the dataset was loaded and visually explored using KDE plots and pair plots to analyze feature distributions and class separability. Next, the data was normalized using Min-Max scaling, and split into training and test sets in an 80/20 ratio. A custom Knn class was developed to handle the core functionality of the algorithm, supporting both Euclidean and Manhattan distance metrics.

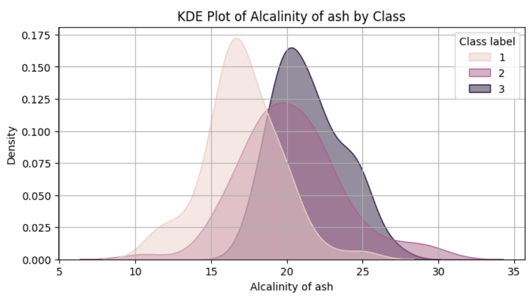
The performance of the model was tested for multiple values of *k* (1, 3, 5, 7, 9, 11, 13), and evaluated using metrics such as accuracy, confusion matrices, and classification reports. The results were visualized through informative plots including Accuracy vs. K and confusion matrix heatmaps, highlighting the effect of different parameter settings on model performance.

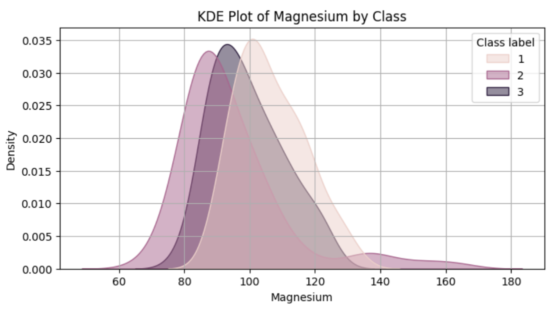
2.KDE and Pair Plots of Features with All Classes

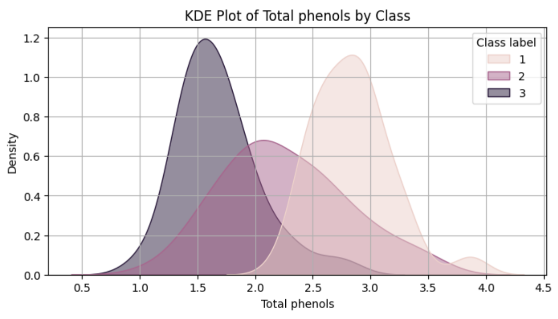


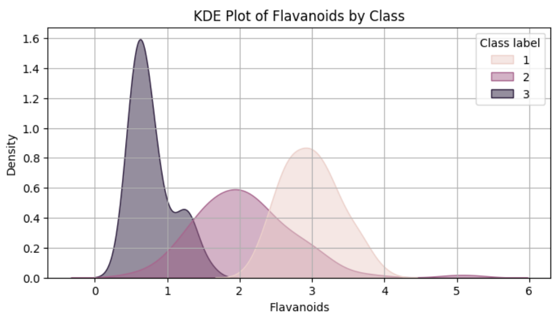


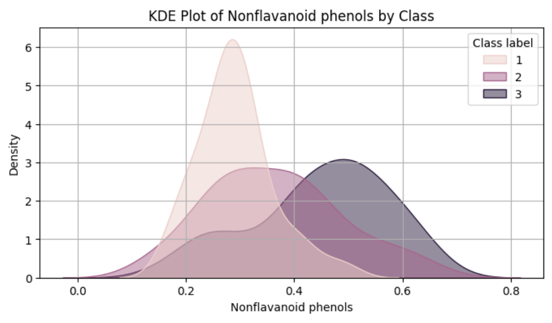


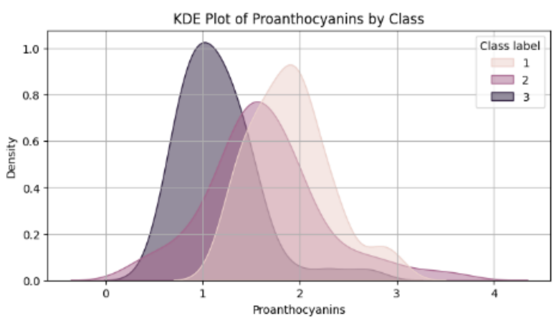


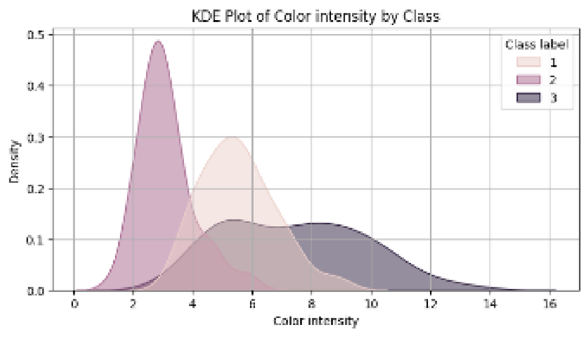


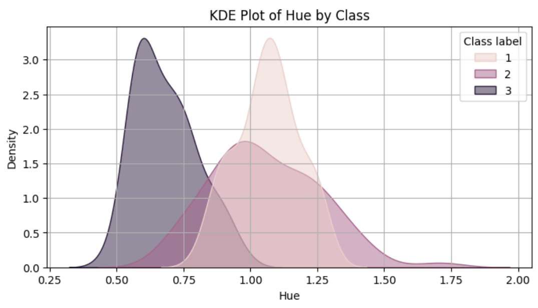


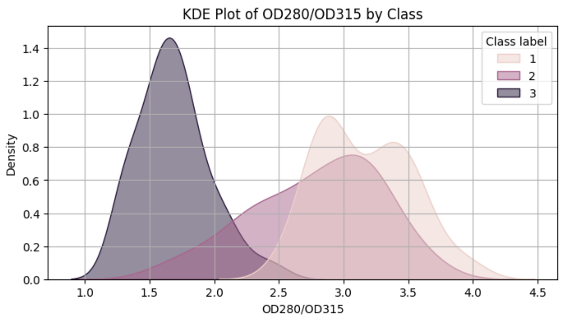


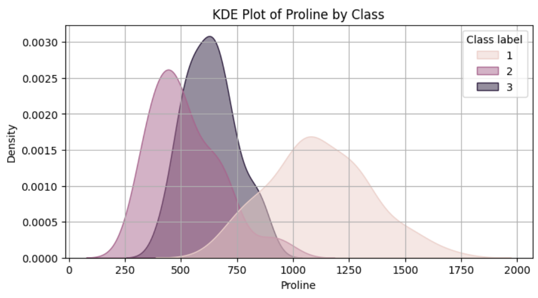






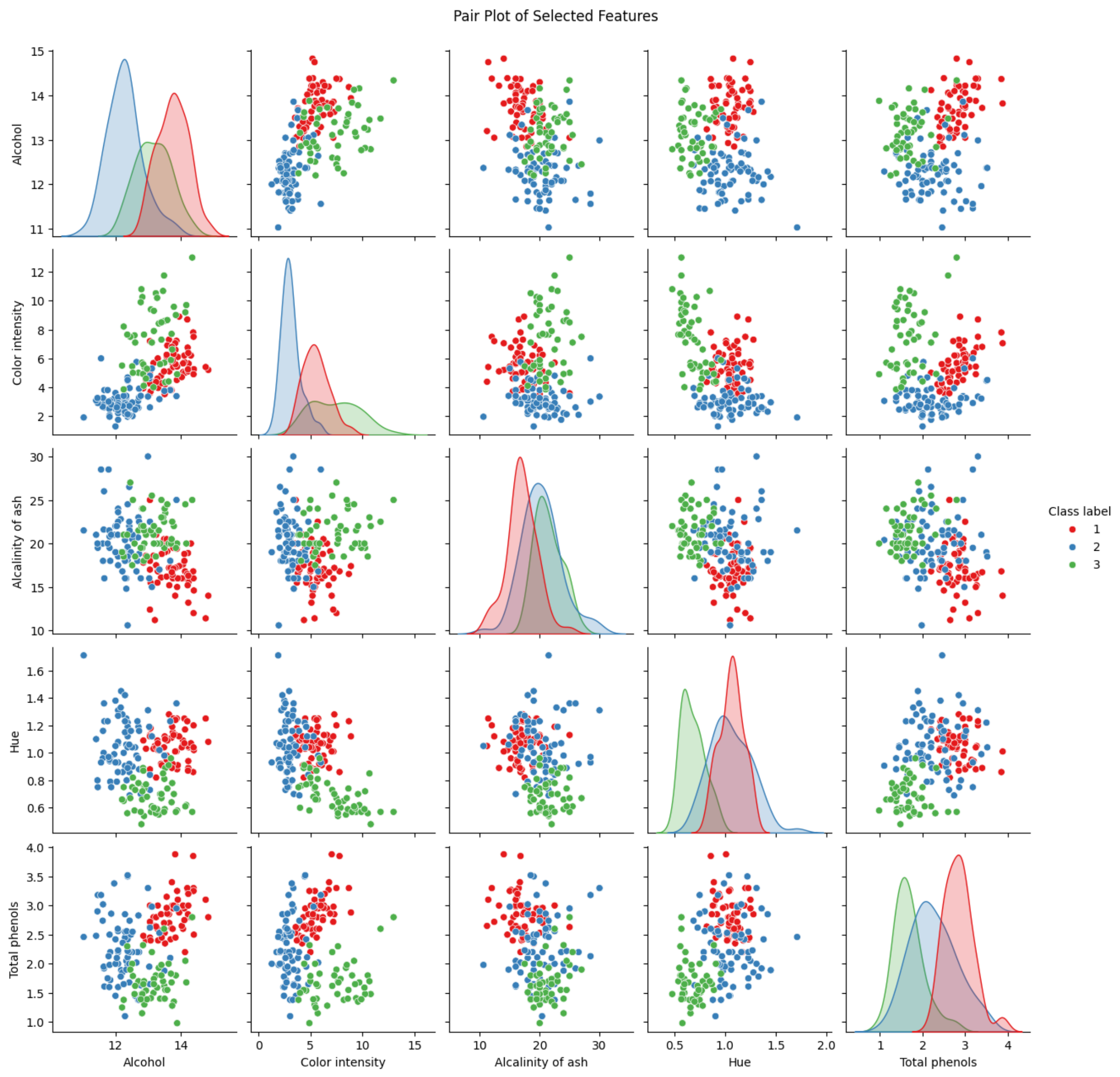






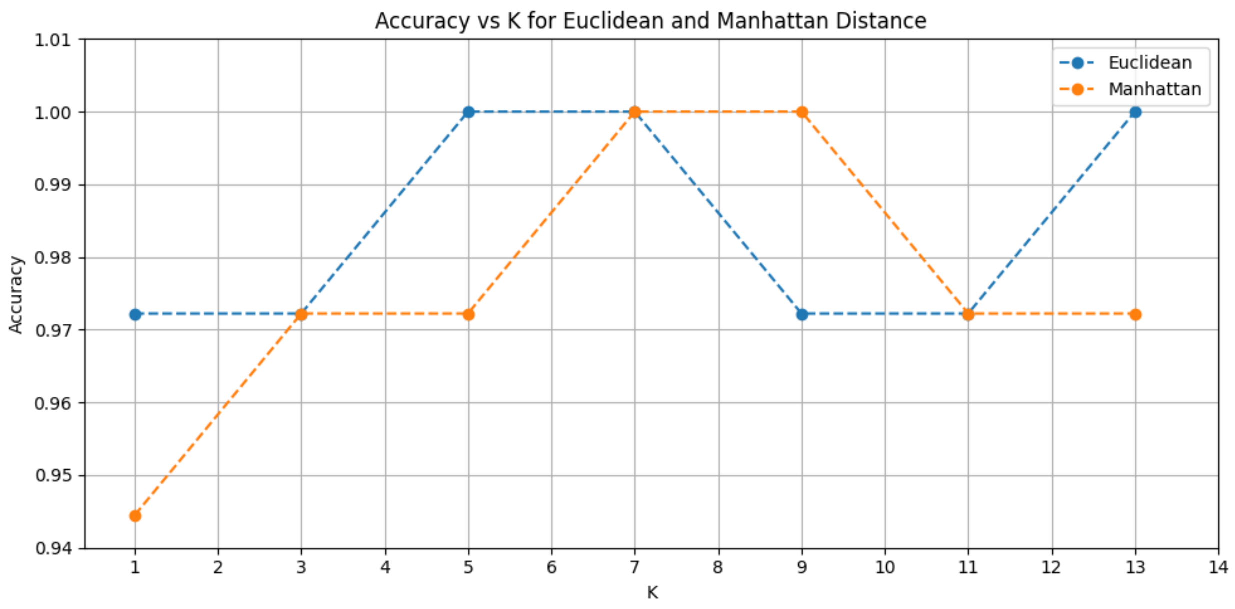
```
import matplotlib.pyplot as plt
import seaborn as sns

feature_names = df.columns[1:]  # Class

for feature in feature_names:
    plt.figure(figsize=(8, 4))
    sns.kdeplot(data=df, x=feature, hue="Class label", fill=True,
common_norm=False, alpha=0.5)
    plt.title(f"KDE Plot of {feature} by Class")
    plt.xlabel(feature)
    plt.ylabel("Density")
    plt.grid(True)
    plt.show()
```

Pair Plot of Selected Features



```
import seaborn as sns
import matplotlib.pyplot as plt

selected_features = ['Alcohol', 'Color intensity', 'Alcalinity of ash',
'Hue', 'Total phenols']

df["Class label"] = df["Class label"].astype(str)

sns.pairplot(df[selected_features + ['Class label']], hue="Class
label", palette="Set1")
plt.suptitle("Pair Plot of Selected Features", y=1.02)
plt.show()
```

Impact of K to Model Performance



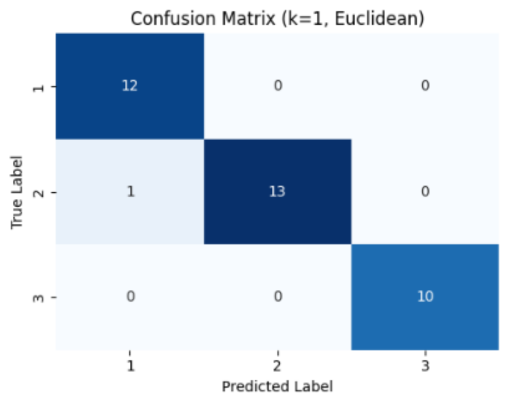
The graph above illustrates how the accuracy of the custom-built k-NN classifier varies with different values of k (1 to 13, odd numbers only) using both Euclidean and Manhattan distance metrics. Both distance metrics perform very well across all tested k values, achieving accuracy scores above 94%, with several instances reaching 100% accuracy. The model reaches its peak performance at k = 5 and k = 7 for both Euclidean and Manhattan distances. This suggests that a moderate value of k provides the best generalization on the test set.

At k = 1, the model is more sensitive to noise, especially with the Manhattan distance where accuracy dips slightly.
At higher values like k = 11 or 13, accuracy decreases again for both metrics. This is likely due to the model becoming too "smooth" and less sensitive to local variations.

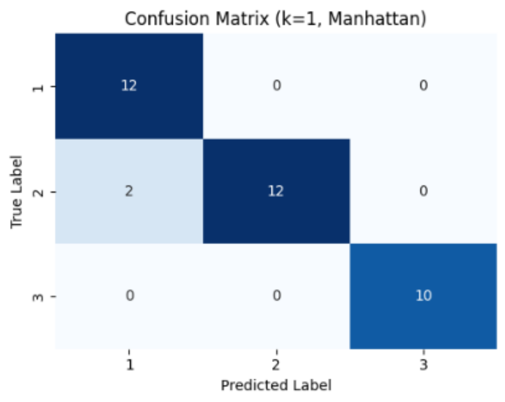
Manhattan distance shows more stable performance across increasing k values, while Euclidean slightly fluctuates.

```
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score

k_values = [1, 3, 5, 7, 9, 11, 13]
x_ticks = list(range(1, 14 + 1))

distance_metrics = ["euclidean", "manhattan"]

plt.figure(figsize=(10, 5))

for metric in distance_metrics:
    accuracies = []
    for k in k_values:
        model = KNN(k=k, distance_metric=metric)
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        acc = accuracy_score(y_test, y_pred)
        accuracies.append(acc)
        print(f"{metric} - k={k} -> Accuracy: {acc:.4f}")

    plt.plot(k_values, accuracies, marker='o', linestyle='--',
label=metric.capitalize())

plt.xticks(x_ticks)

plt.title("Accuracy vs K for Euclidean and Manhattan Distance")
plt.xlabel("K")
plt.ylabel("Accuracy")
plt.ylim(0.94, 1.01)
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```

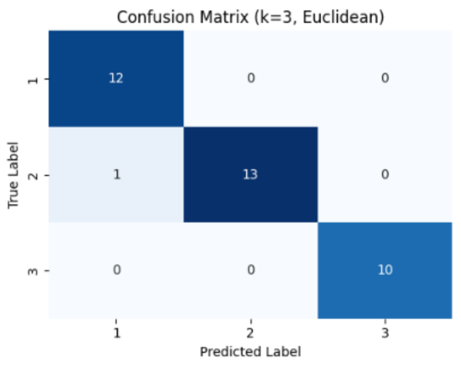
Confusion Matrices



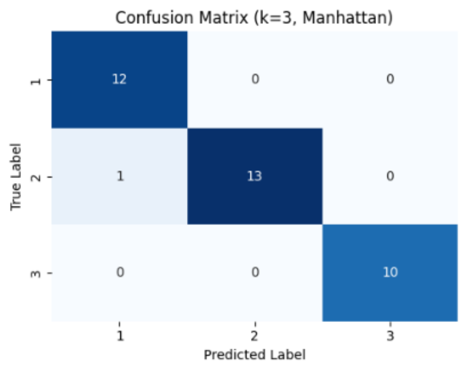
```
Classification Report (k=1, Euclidean):

              precision    recall  f1-score   support

           1       0.92      1.00      0.96        12
           2       1.00      0.93      0.96        14
           3       1.00      1.00      1.00        10

    accuracy                           0.97        36
   macro avg       0.97      0.98      0.97        36
weighted avg       0.97      0.97      0.97        36
```



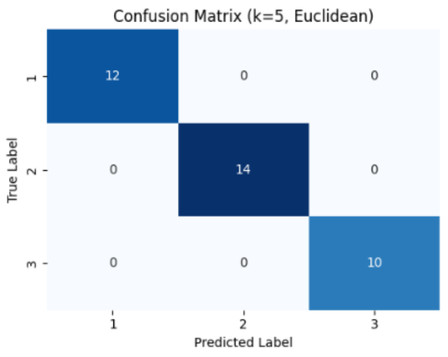
```
Classification Report (k=1, Manhattan):

              precision    recall  f1-score   support

           1       0.86      1.00      0.92        12
           2       1.00      0.86      0.92        14
           3       1.00      1.00      1.00        10

    accuracy                           0.94        36
   macro avg       0.95      0.95      0.95        36
weighted avg       0.95      0.94      0.94        36
```



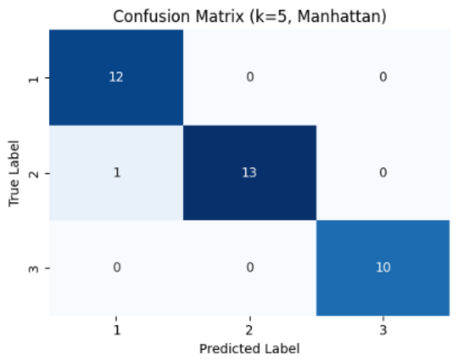
```
Classification Report (k=3, Euclidean):

              precision    recall  f1-score   support

           1       0.92      1.00      0.96        12
           2       1.00      0.93      0.96        14
           3       1.00      1.00      1.00        10

    accuracy                           0.97        36
   macro avg       0.97      0.98      0.97        36
weighted avg       0.97      0.97      0.97        36
```



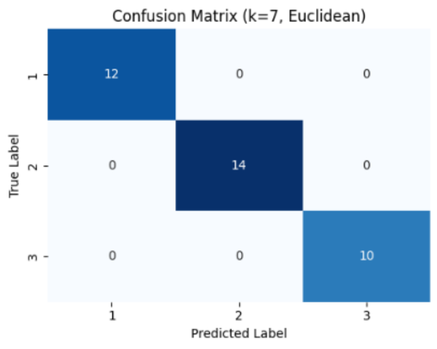
```
Classification Report (k=3, Manhattan):

              precision    recall  f1-score   support

           1       0.92      1.00      0.96        12
           2       1.00      0.93      0.96        14
           3       1.00      1.00      1.00        10

    accuracy                           0.97        36
   macro avg       0.97      0.98      0.97        36
weighted avg       0.97      0.97      0.97        36
```



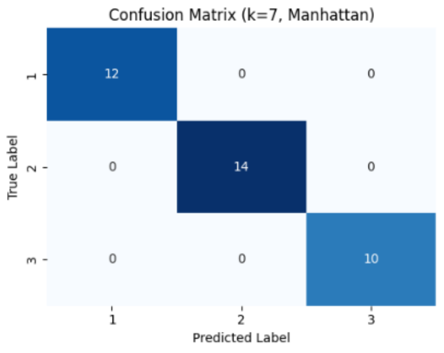
```
Classification Report (k=5, Euclidean):

              precision    recall  f1-score   support

           1       1.00      1.00      1.00        12
           2       1.00      1.00      1.00        14
           3       1.00      1.00      1.00        10

    accuracy                           1.00        36
   macro avg       1.00      1.00      1.00        36
weighted avg       1.00      1.00      1.00        36
```



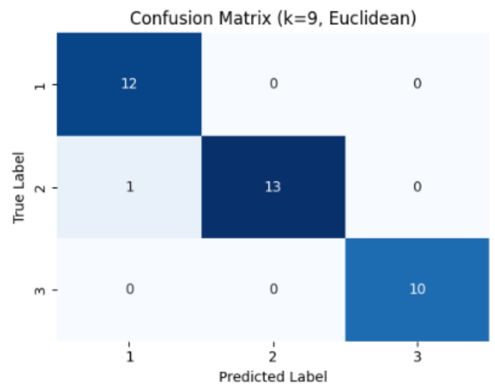
```
Classification Report (k=5, Manhattan):

              precision    recall  f1-score   support

           1       0.92      1.00      0.96        12
           2       1.00      0.93      0.96        14
           3       1.00      1.00      1.00        10

    accuracy                           0.97        36
   macro avg       0.97      0.98      0.97        36
weighted avg       0.97      0.97      0.97        36
```



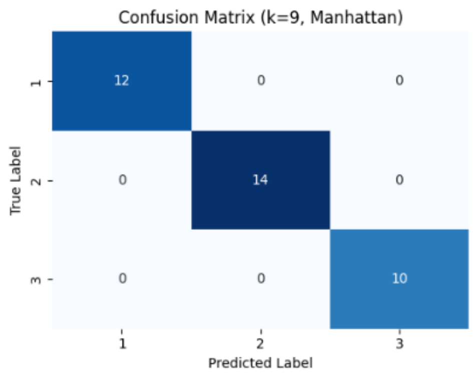
```
Classification Report (k=7, Euclidean):

              precision    recall  f1-score   support

           1       1.00      1.00      1.00        12
           2       1.00      1.00      1.00        14
           3       1.00      1.00      1.00        10

    accuracy                           1.00        36
   macro avg       1.00      1.00      1.00        36
weighted avg       1.00      1.00      1.00        36
```



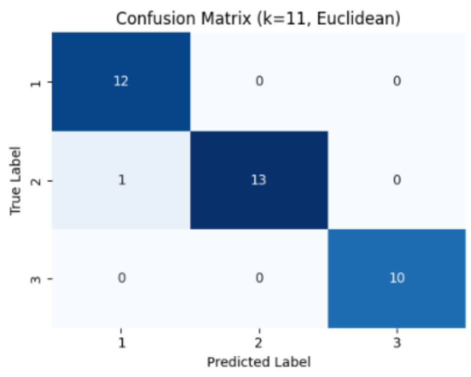
```
Classification Report (k=7, Manhattan):

              precision    recall  f1-score   support

           1       1.00      1.00      1.00        12
           2       1.00      1.00      1.00        14
           3       1.00      1.00      1.00        10

    accuracy                           1.00        36
   macro avg       1.00      1.00      1.00        36
weighted avg       1.00      1.00      1.00        36
```



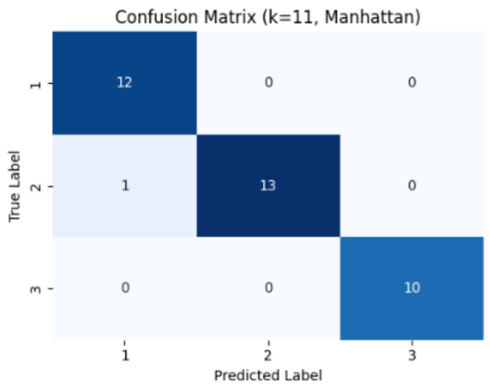
```
Classification Report (k=9, Euclidean):

              precision    recall  f1-score   support

           1       0.92      1.00      0.96        12
           2       1.00      0.93      0.96        14
           3       1.00      1.00      1.00        10

    accuracy                           0.97        36
   macro avg       0.97      0.98      0.97        36
weighted avg       0.97      0.97      0.97        36
```



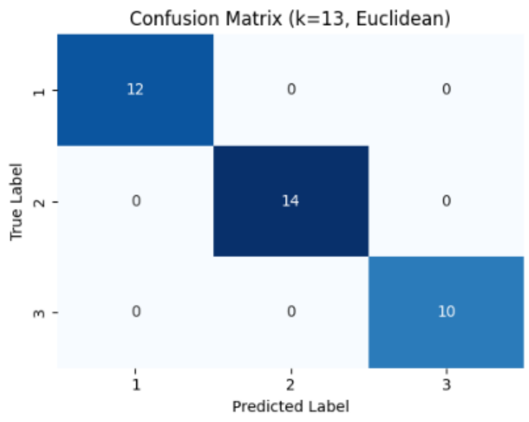
```
Classification Report (k=9, Manhattan):

              precision    recall  f1-score   support

           1       1.00      1.00      1.00        12
           2       1.00      1.00      1.00        14
           3       1.00      1.00      1.00        10

    accuracy                           1.00        36
   macro avg       1.00      1.00      1.00        36
weighted avg       1.00      1.00      1.00        36
```

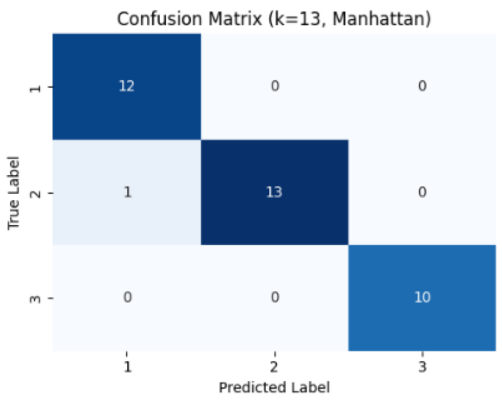


```
Classification Report (k=11, Euclidean):

              precision    recall  f1-score   support

           1       0.92      1.00      0.96        12
           2       1.00      0.93      0.96        14
           3       1.00      1.00      1.00        10

    accuracy                           0.97        36
   macro avg       0.97      0.98      0.97        36
weighted avg       0.97      0.97      0.97        36
```



```
Classification Report (k=11, Manhattan):

              precision    recall  f1-score   support

           1       0.92      1.00      0.96        12
           2       1.00      0.93      0.96        14
           3       1.00      1.00      1.00        10

    accuracy                           0.97        36
   macro avg       0.97      0.98      0.97        36
weighted avg       0.97      0.97      0.97        36
```





```
Classification Report (k=13, Euclidean):

              precision    recall  f1-score   support

           1       1.00      1.00      1.00        12
           2       1.00      1.00      1.00        14
           3       1.00      1.00      1.00        10

    accuracy                           1.00        36
   macro avg       1.00      1.00      1.00        36
weighted avg       1.00      1.00      1.00        36
```

```
Classification Report (k=13, Manhattan):

              precision    recall  f1-score   support

           1       0.92      1.00      0.96        12
           2       1.00      0.93      0.96        14
           3       1.00      1.00      1.00        10

    accuracy                           0.97        36
   macro avg       0.97      0.98      0.97        36
weighted avg       0.97      0.97      0.97        36
```

Whole Code

```
# Import required libraries
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report
import matplotlib.pyplot as plt
import seaborn as sns
from collections import Counter

# Load the Wine dataset
columns = [
    'Class label', 'Alcohol', 'Malic acid', 'Ash', 'Alcalinity of ash',
    'Magnesium', 'Total phenols', 'Flavanoids', 'Nonflavanoid phenols',
    'Proanthocyanins', 'Color intensity', 'Hue', 'OD280/OD315',
'Proline'
]

df = pd.read_csv('wine.data', header=None, names=columns)

# Normalize the features using Min-Max Scaling
X = df.drop("Class label", axis=1)
y = df["Class label"]

scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)
```

```python
# Split into training and testing sets (80/20)
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42, stratify=y
)

# Define distance functions
def euclidean_distance(x1, x2):
    return np.sqrt(np.sum((x1 - x2) ** 2))

def manhattan_distance(x1, x2):
    return np.sum(np.abs(x1 - x2))

# Define the KNN class
class KNN:
    def __init__(self, k=3, distance_metric="euclidean"):
        self.k = k
        self.distance_metric = distance_metric

    def fit(self, X, y):
        self.X_train = X
        self.y_train = np.array(y)  # Ensure indexing works properly

    def _compute_distance(self, x1, x2):
        if self.distance_metric == "euclidean":
            return euclidean_distance(x1, x2)
        elif self.distance_metric == "manhattan":
            return manhattan_distance(x1, x2)
        else:
            raise ValueError("Unsupported distance metric")

    def _predict_single(self, x):
        distances = [self._compute_distance(x, x_train) for x_train in
self.X_train]
        k_indices = np.argsort(distances)[:self.k]
        k_labels = [self.y_train[i] for i in k_indices]
        most_common = Counter(k_labels).most_common(1)
        return most_common[0][0]

    def predict(self, X):
        return np.array([self._predict_single(x) for x in X])

# Analyze accuracy across different k values and metrics
k_values = [1, 3, 5, 7, 9, 11, 13]
distance_metrics = ["euclidean", "manhattan"]

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

for metric in distance_metrics:
    accuracies = []
    for k in k_values:
        model = KNN(k=k, distance_metric=metric)
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        acc = accuracy_score(y_test, y_pred)
        accuracies.append(acc)
        print(f"{metric.capitalize()} - k={k} -> Accuracy: {acc:.4f}")

    # Plot accuracy for this distance metric
    plt.plot(k_values, accuracies, marker='o', linestyle='--',
label=metric.capitalize())
    for k, acc in zip(k_values, accuracies):
        plt.text(k, acc + 0.003, f"{acc:.2f}", ha='center', fontsize=9)

# Format the plot
plt.title("Accuracy vs K for Euclidean and Manhattan Distance")
plt.xlabel("K")
plt.ylabel("Accuracy")
plt.xticks(range(1, 14 + 1))  # Ensure axis ticks show 1 through 13
plt.ylim(0.94, 1.01)
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

# Confusion matrices and classification reports for all settings
class_labels = sorted(y.unique())

for metric in distance_metrics:
    print(f"\n===== Distance Metric: {metric.upper()} =====\n")
    for k in k_values:
        model = KNN(k=k, distance_metric=metric)
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)

        # Confusion Matrix
        cm = confusion_matrix(y_test, y_pred, labels=class_labels)
        plt.figure(figsize=(5, 4))
        sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                    xticklabels=class_labels, yticklabels=class_labels,
cbar=False)
        plt.title(f'Confusion Matrix (k={k}, {metric.capitalize()})')
        plt.xlabel('Predicted Label')
        plt.ylabel('True Label')
        plt.tight_layout()
        plt.show()

        # Classification Report
        print(f"Classification Report (k={k},
{metric.capitalize()}):\n")
        print(classification_report(y_test, y_pred, digits=2))
        print("-" * 60)
```

Comments and Conclusion

The results showed that the model consistently achieved high accuracy scores, often exceeding 97%, with several configurations reaching 100% accuracy. The best overall performance was typically observed at k = 5 or k = 7, regardless of the distance metric used. This supports the common understanding that very small values of k can cause overfitting, while larger values may smooth the decision boundaries too much, reducing the model's sensitivity.

The comparison between distance metrics revealed that Euclidean distance performed slightly better in some cases, while Manhattan distance exhibited more consistent accuracy across increasing k values. However, the performance difference between the two was relatively minor.

Visual analyses using KDE plots, pair plots, confusion matrices, and classification reports all provided deeper insights into the model's behavior, showing which features helped distinguish the classes and where misclassifications tended to occur.

GitHub Repository

All code and Jupyter notebooks can be found at:
https://github.com/HamzaEmirTarcan/ELE489-HW1