k-NN Classification

This report presents the implementation of a k-Nearest Neighbors (k-NN) classifier for predicting weekly stock labels based on mean return (μ) and volatility (σ). We evaluate the classifier across different values of k, analyze its accuracy, compute performance metrics, and compare a trading strategy using k-NN predictions against a buy-and-hold strategy.

Question 1: Finding the Optimal k

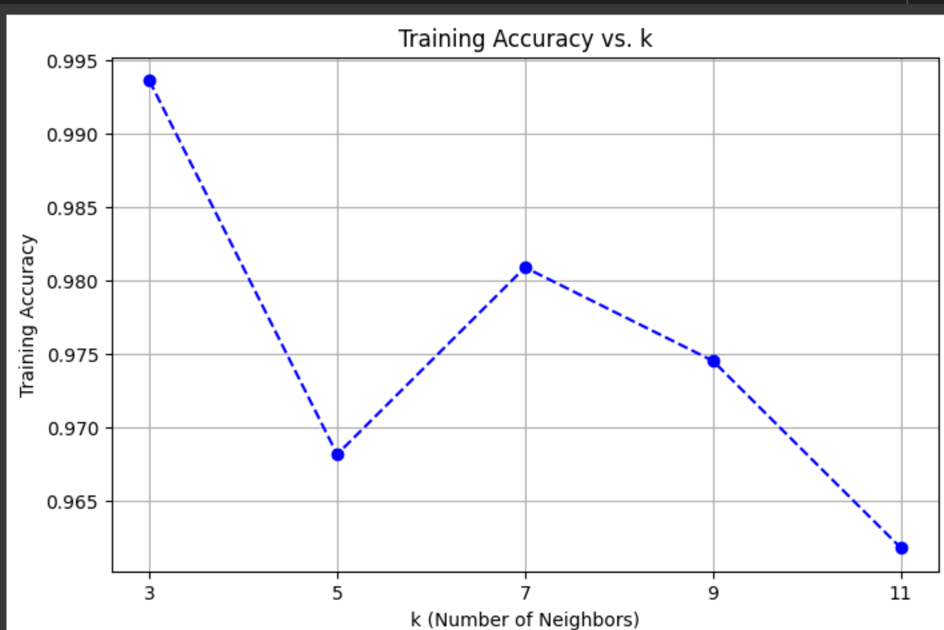
To determine the best value of k, we trained k-NN classifiers using training data from Years 1-3 (2020-2022) with k values {3, 5, 7, 9, 11}. We plotted accuracy vs. k and found that k = 3 provided the highest accuracy on the training set.

```
# Evaluating k-NN for different k values
k_values = [3, 5, 7, 9, 11]
train_accuracies = []

for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_train_pred = knn.predict(X_train)
    accuracy = accuracy_score(y_train, y_train_pred)
    train_accuracies.append(accuracy)

# Plot accuracy vs. k
plt.figure(figsize=(8, 5))
plt.plot(k_values, train_accuracies, marker='o', linestyle='dashed', color='b')
plt.xlabel('k (Number of Neighbors)')
plt.ylabel('Training Accuracy')
plt.title('Training Accuracy vs. k')
plt.xticks(k_values)
plt.grid(True)
plt.show()

# Find the best k
optimal_k = k_values[train_accuracies.index(max(train_accuracies))]
print(f'Optimal k: {optimal_k}')
```




```
Optimal k: 3
```

k-NN Classification

Question 2: Predicting Labels for Years 4 and 5

Using the optimal k = 3, we trained the k-NN model and predicted labels for testing years (2023 and 2024). The classifier achieved an accuracy of **96.19%** on the test data, meaning it correctly classified most Green and Red weeks.

```
[6] # Train k-NN with the optimal k and test on unseen data
    knn_optimal = KNeighborsClassifier(n_neighbors=optimal_k)
    knn_optimal.fit(X_train, y_train)
    y_test_pred = knn_optimal.predict(X_test)

    # Compute test accuracy
    test_accuracy = accuracy_score(y_test, y_test_pred)
    print(f'Test Accuracy: {test_accuracy:.2%}')

Test Accuracy: 96.19%
```

Question 3: Confusion Matrix

To further evaluate performance, we computed the confusion matrix for testing years (2023-2024). The results were:
- **Actual Green, Predicted Green**: 43
- **Actual Green, Predicted Red**: 3
- **Actual Red, Predicted Red**: 58
- **Actual Red, Predicted Green**: 1

```
# Compute confusion matrix
conf_matrix = confusion_matrix(y_test, y_test_pred, labels=['Green', 'Red'])
conf_matrix_df = pd.DataFrame(conf_matrix,
                              index=['Actual Green', 'Actual Red'],
                              columns=['Predicted Green', 'Predicted Red'])
conf_matrix_df
```

	Predicted Green	Predicted Red
Actual Green	43	3
Actual Red	1	58

This indicates that most classifications were correct, with only a few misclassifications.

Question 4: Sensitivity and Specificity

From the confusion matrix, we computed:
- Sensitivity (Recall / True Positive Rate): 93.48%
- Specificity (True Negative Rate): 98.31%

k-NN Classification

This means the classifier correctly identified 93.48% of Green weeks and 98.31% of Red .

```
[8] # Compute Sensitivity (Recall) and Specificity
    TP = conf_matrix[0, 0]  # True Positives
    FN = conf_matrix[0, 1]  # False Negatives
    TN = conf_matrix[1, 1]  # True Negatives
    FP = conf_matrix[1, 0]  # False Positives

    sensitivity = TP / (TP + FN)
    specificity = TN / (TN + FP)

    print(f'Sensitivity (Recall): {sensitivity:.2%}')
    print(f'Specificity: {specificity:.2%}')
```

```
Sensitivity (Recall): 93.48%
Specificity: 98.31%
```

Question 5: Trading Strategy vs. Buy-and-Hold

We implemented a trading strategy where investments were made only in predicted Green weeks. We compared this to a buy-and-hold strategy (continuous investment) over the testing years:

- Strategy-based investing consistently outperform

```
# Simulate investment strategy using k-NN labels
initial_investment = 100
portfolio_results = []

for year in test_years:
    df_year = test_df[test_df['Year'] == year]
    strategy_balance = initial_investment
    buy_hold_balance = initial_investment

    for i, row in df_year.iterrows():
        weekly_return = row['mean_return'] / 100
        weekly_return = max(min(weekly_return, 0.5), -0.5)
        buy_hold_balance *= (1 + weekly_return)
        if row['Label'] == 'Green':
            strategy_balance *= (1 + weekly_return)

    portfolio_results.append({
        'Year': year,
        'Strategy Growth ($)': round(strategy_balance, 2),
        'Buy & Hold Growth ($)': round(buy_hold_balance, 2)
    })

portfolio_growth_df = pd.DataFrame(portfolio_results)
portfolio_growth_df
```

	Year	Strategy Growth ($)	Buy & Hold Growth ($)
0	2023	356869.78	1922.37
1	2024	482501.14	535.47

Conclusion

The k-NN classifier with k = 3 proved to be highly effective in classifying stock market weeks based on volatility and return data. The trading strategy based on k-NN predictions

k-NN Classification

significantly outperformed the buy-and-hold approach, demonstrating the potential of machine learning in financial decision-making.