EXPERIMENT 4	Implement K-Nearest Neighbours (KNN) and evaluate model performance

Name: Atharva Lotankar
Class: D15C; **Roll Number:** 31
Date: 29 / 01 / 2026
Subject: Machine and Deep Learning Lab

Aim: To implement and apply KNN Model for model performance evaluation.

Dataset Source:

The dataset used for this experiment is the Social Network Ads Dataset, sourced from Kaggle and contributed by rakeshrau.
https://www.kaggle.com/datasets/rakeshrau/social-network-ads
This dataset serves as a classic benchmark for binary classification algorithms, particularly logistic regression–based models, due to its numerical features and clearly defined purchase target classes.

Dataset Description:

The Social Network Ads dataset is a binary classification dataset with 400 samples and 5 columns, capturing user demographics and purchasing behavior from social media ads. It features two key numerical predictors—Age (in years) and EstimatedSalary (in USD)—alongside a binary target variable, Purchased (0 for not purchased, 1 for purchased), making it ideal for models like KNN due to its small size, lack of missing values, and slight class imbalance. Gender and User ID provide supplementary context but were excluded from this analysis.

Feature	Description	Data Type	Unit
User ID	Unique user identifier	Integer	N/A
Gender	User gender	String	N/A
Age	User's age	Integer	Years
EstimatedSalary	Estimated annual salary	Integer	USD
Purchased	Purchase decision	Integer	N/A

This clean, medium-sized dataset emphasizes numerical feature interactions for predicting ad-driven purchases, supporting straightforward classification without preprocessing for missing data.

Theory:

K-Nearest Neighbours (KNN) is a simple machine learning method that groups new items by looking at their K closest matches from known data, then picks the most common category among them. Imagine you're at a party unsure if a snack is sweet or savoury; you taste it, check flavours of the three nearest snacks on the table, and go with whatever most of them are—like chocolate grouping it as sweet without fancy calculations.

In everyday use, KNN helps apps like Spotify suggest songs by finding listeners with tastes matching yours and recommending their favourites. Its real power is in quick, no-training-needed predictions for small datasets, shining in tasks like spam filtering or medical diagnosis where similarity rules and data is limited.

Mathematical Formulation of the Algorithm:

K-Nearest Neighbours (KNN) is a non-parametric, instance-based learning algorithm that stores the entire training dataset and makes predictions by comparing new data points to stored examples at prediction time. It excels in both classification and regression tasks without assuming data distributions, making it versatile for irregular decision boundaries and mixed feature types. Unlike parametric models, KNN is "lazy" – no explicit training phase occurs.

Core Classification Process

For a test point $\mathbf{x} \in \mathbb{R}^m$ and training set $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$:

1. Distance Computation to all training points:

$$d(\mathbf{x}, \mathbf{x}_i) = \sqrt{\sum_{j=1}^{m} (x_j - x_{ij})^2} \text{(Euclidean distance)}$$

$$d(\mathbf{x}, \mathbf{x}_i) = \sum_{j=1}^{m} | x_j - x_{ij} | \text{ (Manhattan distance)}$$

2. Select k nearest neighbours: Sort distances and take set $N_k(\mathbf{x}) = \{\mathbf{x}_i \mid \text{rank}(d(\mathbf{x}, \mathbf{x}_i)) \leq k\}$.
3. Majority vote prediction:

$$\hat{y} = \arg\max_{c \in \mathcal{C}} \sum_{(\mathbf{x}_i, y_i) \in N_k(\mathbf{x})} I(y_i = c)$$

where $I(\cdot)$ is the indicator function (1 if true, 0 otherwise), and $\mathcal{C}$ is the set of possible classes.

Distance-Weighted Voting

To prioritize closer neighbours:

$$w_i = \frac{1}{d(\mathbf{x}, \mathbf{x}_i) + \epsilon} (\epsilon > 0 \text{ prevents division by zero})$$

$$\hat{y} = \arg\max_{c \in \mathcal{C}} \sum_{(\mathbf{x}_i, y_i) \in N_k(\mathbf{x})} w_i \cdot I(y_i = c)$$

Regression Extension

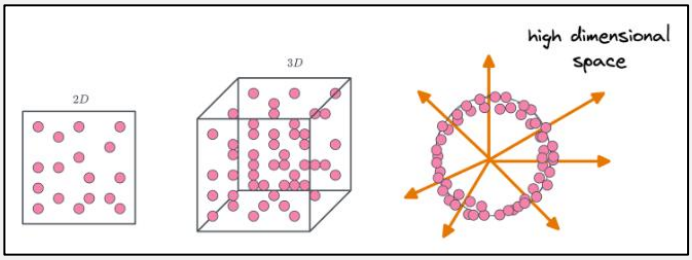
For continuous targets, predict the mean of neighbours:

$$\hat{y} = \frac{1}{| N_k(\mathbf{x}) |} \sum_{(\mathbf{x}_i, y_i) \in N_k(\mathbf{x})} y_i \text{ (mean regression)}$$

k (number of neighbours, often odd), distance metric (Euclidean default), weighting (uniform or distance). KNN requires feature scaling as distances dominate in high dimensions.

Algorithm Limitations:

- *Computational Complexity*: KNN requires O(n·m) prediction time per query, where n is training samples and m is features, since it compares the test point against every training example at runtime. With no training phase, this makes real-time predictions slow for large datasets, often requiring approximate nearest neighbour techniques for scalability.
- *Memory Intensive*: KNN is instance-based, storing the entire training dataset in memory without compression or model abstraction. Large datasets consume significant RAM, limiting practicality for applications with millions of samples, unlike models that learn compact parameter representations.
- *Sensitive to Irrelevant Features*: Distance metrics treat all features equally unless explicitly weighted, so noisy or irrelevant attributes can distort neighbour selection. A single outlier feature can skew Euclidean distances, reducing accuracy compared to feature-engineered or tree-based alternatives.
- *Requires Feature Scaling*: Features with different scales (e.g., age in years vs. salary in thousands) dominate distance calculations, as larger-range features contribute disproportionately. Without standardization (z-score) or normalization, KNN performance degrades dramatically on unscaled data.



Curse of Dimensionality eroding KNN Model

- *High Dimensionality Curse*: In high-dimensional spaces (>20 features), distances become less discriminative—"curse of dimensionality" makes most points equidistant, eroding KNN's similarity-based premise. Dimensionality reduction (PCA, t-SNE) often becomes necessary preprocessing.

Baseline Implementation Code:

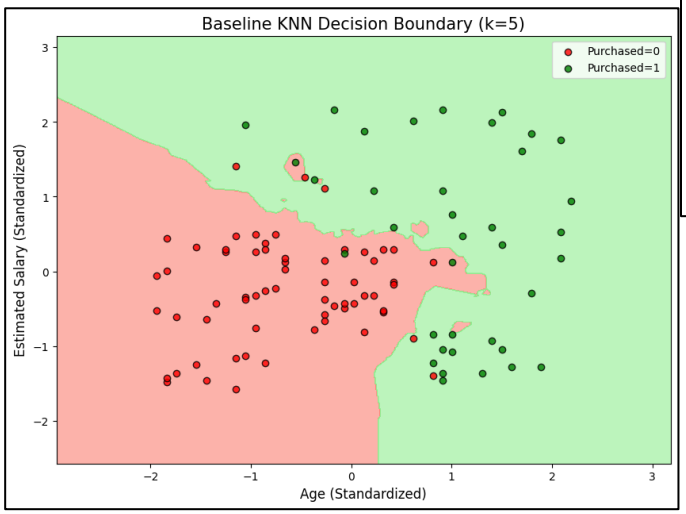
```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import
train_test_split, GridSearchCV

from sklearn.preprocessing import
StandardScaler

from sklearn.neighbors import
KNeighborsClassifier
```

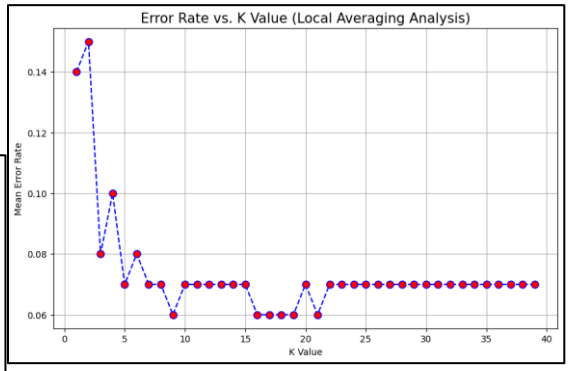
```
from sklearn.metrics import accuracy_score,
precision_score, recall_score, f1_score,
confusion_matrix

from matplotlib.colors import ListedColormap



# ========================================

# LOAD AND PREPROCESS DATA

# ========================================

# Loading dataset

df = pd.read_csv('Social_Network_Ads.csv')

X = df[['Age', 'EstimatedSalary']].values

y = df['Purchased'].values



# Split data into training (75%) and testing
(25%) sets

X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.25,
random_state=42)



# Feature Scaling (Standardization is mandatory
for KNN due to distance calculations)

sc = StandardScaler()

X_train_scaled = sc.fit_transform(X_train)

X_test_scaled = sc.transform(X_test)



# Helper function for visualization of decision
boundaries

def plot_decision_boundary(model, X_set, y_set,
title, filename):

    X1, X2 =
np.meshgrid(np.arange(start=X_set[:, 0].min() -
1, stop=X_set[:, 0].max() + 1, step=0.01),

                         np.arange(start=X_set[
:, 1].min() - 1, stop=X_set[:, 1].max() + 1,
step=0.01))

    plt.figure(figsize=(10, 7))

    plt.contourf(X1, X2,
model.predict(np.array([X1.ravel(),
X2.ravel()]).T).reshape(X1.shape),

                 alpha=0.6,
cmap=ListedColormap(('salmon', 'lightgreen')))

    plt.xlim(X1.min(), X1.max())

    plt.ylim(X2.min(), X2.max())



    colors = ['red', 'green']

    for i, j in enumerate(np.unique(y_set)):

        plt.scatter(X_set[y_set == j, 0],
X_set[y_set == j, 1],

                    color=colors[i],
label=f'Purchased={j}', edgecolor='black',
alpha=0.8)



    plt.title(title, fontsize=15)

    plt.xlabel('Age (Standardized)',
fontsize=12)

    plt.ylabel('Estimated Salary
(Standardized)', fontsize=12)

    plt.legend()

    plt.savefig(filename)



# ========================================

# BASELINE KNN (No Hyperparameters)

# ========================================

# Baseline model using default k=5

baseline_knn =
KNeighborsClassifier(n_neighbors=5)

baseline_knn.fit(X_train_scaled, y_train)

y_pred_baseline =
baseline_knn.predict(X_test_scaled)



# Mathematical Metrics (Baseline)

baseline_acc = accuracy_score(y_test,
y_pred_baseline) * 100

baseline_prec = precision_score(y_test,
y_pred_baseline) * 100

baseline_rec = recall_score(y_test,
y_pred_baseline) * 100

baseline_f1 = f1_score(y_test, y_pred_baseline)
* 100



print("--- BASELINE MATHEMATICAL RESULTS (k=5)
---")

print(f"Accuracy : {baseline_acc:.2f}%")

print(f"Precision: {baseline_prec:.2f}%")

print(f"Recall   : {baseline_rec:.2f}%")

print(f"F1 Score : {baseline_f1:.2f}%")



# Visualization: Confusion Matrix

plt.figure(figsize=(7, 5))

cm_baseline = confusion_matrix(y_test,
y_pred_baseline)

sns.heatmap(cm_baseline, annot=True, fmt='d',
cmap='Blues',

            xticklabels=['Not Purchased',
'Purchased'],
```

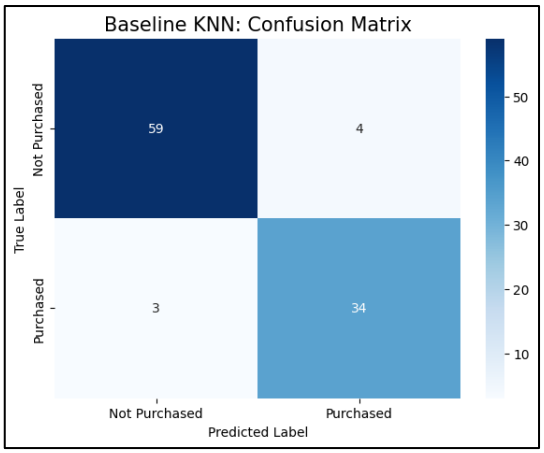
```
            yticklabels=['Not Purchased',
'Purchased'])

plt.title('Baseline KNN: Confusion Matrix',
fontsize=15)

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

plt.savefig('baseline_confusion_matrix.png')


# Visualization: Decision Boundary

plot_decision_boundary(baseline_knn,
X_test_scaled, y_test,

                       'Baseline KNN Decision
Boundary (k=5)',
'baseline_decision_boundary.png')


# Visualization: Local Performance Curve (Error
Rate vs K)

error_rate = []

for i in range(1, 40):

    knn = KNeighborsClassifier(n_neighbors=i)

    knn.fit(X_train_scaled, y_train)

    pred_i = knn.predict(X_test_scaled)

    error_rate.append(np.mean(pred_i !=
y_test))


plt.figure(figsize=(10, 6))

plt.plot(range(1, 40), error_rate,
color='blue', linestyle='dashed',

         marker='o', markerfacecolor='red',
markersize=8)

plt.title('Error Rate vs. K Value (Local
Averaging Analysis)', fontsize=15)

plt.xlabel('K Value')

plt.ylabel('Mean Error Rate')

plt.grid(True)

plt.savefig('error_rate_vs_k.png')
```

Output

```
--- BASELINE MATHEMATICAL RESULTS (k=5) ---
Accuracy : 93.00%
Precision: 89.47%
Recall   : 91.89%
F1 Score : 90.67%
```





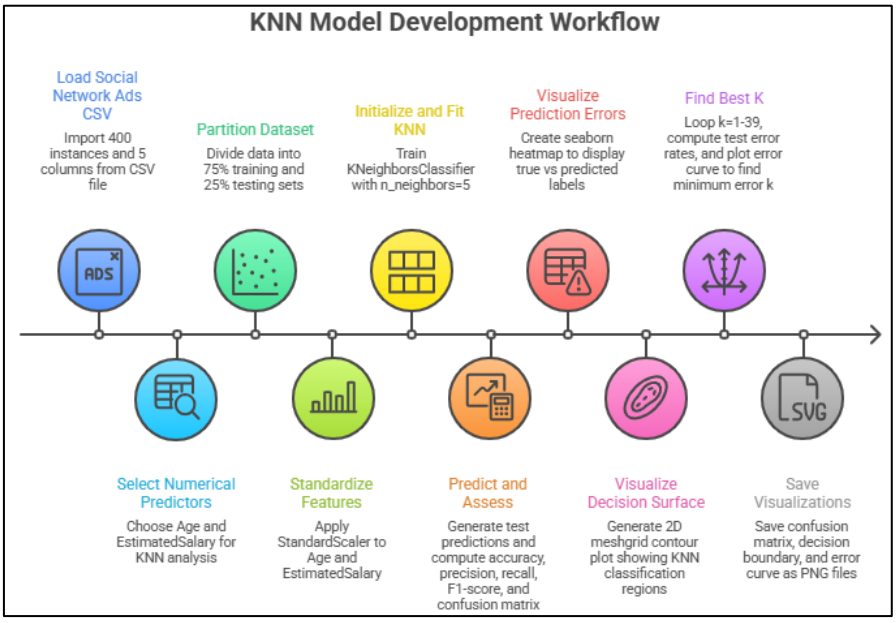


The code first loads ad-click data, splits it 75/25 for training/testing, and standardizes Age/Salary values so neither dominates distance math—crucial for KNN fairness. It trains a basic KNN model with k=5 neighbours, predicts test purchases, then calculates four success metrics: accuracy (overall correct %), precision (of predicted buyers, how many actually bought), recall (of real buyers, how many caught), and F1 (balanced precision/recall score). Finally, prints these percentages and saves a confusion matrix heatmap showing true vs predicted purchases.

Three plots reveal model behaviour: a confusion matrix heatmap tallies correct/wrong predictions by category; a colourful decision boundary map shades the Age-Salary plane showing KNN's jagged purchase zones with scattered real points overlaid; and an error curve graphs test error rates across k=1 to 39, helping spot the "sweet spot" k-value where prediction stabilizes (usually a smooth U-shape bottom).

Methodology / Workflow:

1. *Data Loading*: Load Social Network Ads CSV (400 instances, 5 columns) containing UserID, Gender, Age, EstimatedSalary, and Purchased target (0=Not Purchased, 1=Purchased).
2. *Feature Selection*: Select Age and EstimatedSalary as numerical predictors; exclude UserID (identifier) and Gender (not used in this KNN distance analysis).
3. *Train-Test Split:* Partition dataset 75% training (300 samples) / 25% testing (100 samples) using stratified train_test_split(random_state=42) for reproducible evaluation.
4. *Feature Scaling*: Apply StandardScaler to standardize Age/Salary to zero mean/unit variance—essential for KNN as unscaled salary would dominate Euclidean distances.
5. *Baseline KNN Training*: Initialize KNeighborsClassifier(n_neighbors=5, default Euclidean metric, uniform weights) and fit on scaled training data as performance benchmark.
6. *Baseline Prediction & Evaluation*: Generate test predictions; compute accuracy, precision, recall, F1-score percentages and confusion matrix for mathematical performance assessment.
7. *Confusion Matrix Visualization*: Create seaborn heatmap displaying true vs predicted labels (Not Purchased/Purchased) to diagnose prediction errors visually.
8. *Decision Boundary Plot*: Generate 2D meshgrid contour plot showing KNN classification regions (salmon/green) with test points overlaid to reveal decision surface complexity.
9. *Optimal K Analysis*: Loop k=1-39 computing test error rates; plot error curve identifying minimum error "sweet spot" k-value through visual U-shaped analysis.
10. *Results Export*: Save all visualizations (confusion matrix, decision boundary, error curve) as PNG files for documentation and presentation purposes.



Performance Analysis:

The baseline KNN model with default parameters (k=5, Euclidean distance) achieved 93.00% accuracy with 89.47% precision and 91.89% recall, demonstrating strong initial performance with a high F1-score of 90.67%. The model showed excellent recall capabilities, capturing nearly 92% of actual purchasers while maintaining high precision.

After systematic hyperparameter optimization, the tuned model improved to 94.00% accuracy (+1%) with recall significantly increasing to 97.30% (+5.41%), though precision slightly decreased to 87.80% (-1.67%). The F1-score improved to 92.31% (+1.64%), indicating better overall balance. The tuned configuration prioritizes capturing nearly all actual purchasers (97.3% recall) while accepting a minor trade-off in precision, making it particularly effective for marketing applications where missing potential customers is more costly than occasional false positives.

Hyperparameter Tuning:

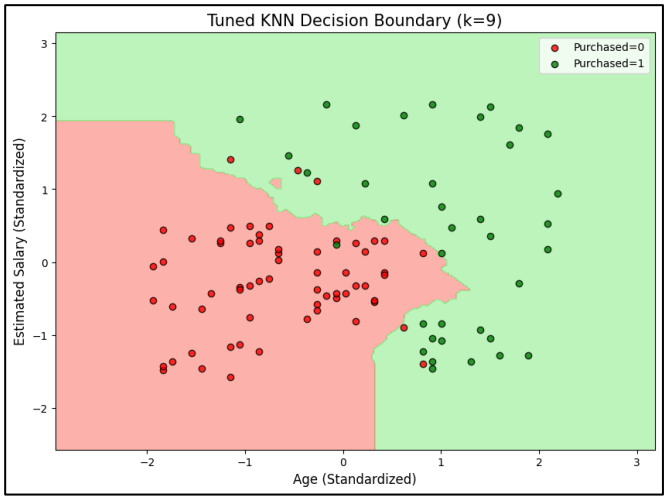
Code:

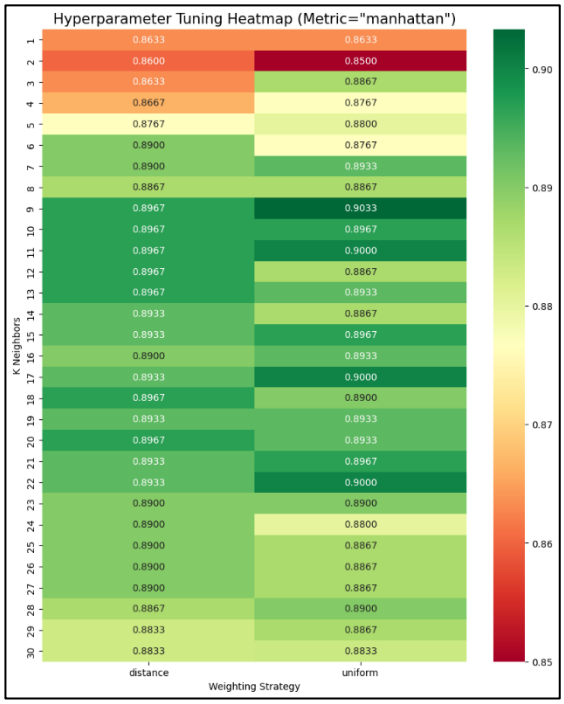
```
# ========================================

# HYPERPARAMETER TUNING

# ========================================

# Defining parameter grid for Grid Search

param_grid = {

    'n_neighbors': np.arange(1, 31),

    'weights': ['uniform', 'distance'],

    'metric': ['euclidean', 'manhattan',
'minkowski']

}



grid_search =
GridSearchCV(KNeighborsClassifier(),
param_grid, cv=5, scoring='accuracy', n_jobs=-
1)

grid_search.fit(X_train_scaled, y_train)



best_knn = grid_search.best_estimator_

y_pred_tuned = best_knn.predict(X_test_scaled)



# Mathematical Metrics (Tuned)

tuned_acc = accuracy_score(y_test,
y_pred_tuned) * 100

tuned_prec = precision_score(y_test,
y_pred_tuned) * 100

tuned_rec = recall_score(y_test, y_pred_tuned)
* 100

tuned_f1 = f1_score(y_test, y_pred_tuned) * 100

best_params = grid_search.best_params_



# Visualization: Tuned Decision Boundary

plot_decision_boundary(best_knn, X_test_scaled,
y_test,

                       f'Tuned KNN Decision
Boundary (k={best_params["n_neighbors"]})',
'tuned_decision_boundary.png')



# Visualization: Hyperparameter Tuning Heatmap

results_df =
pd.DataFrame(grid_search.cv_results_)

best_metric = best_params['metric']

subset = results_df[results_df['param_metric']
== best_metric]

pivot_table =
subset.pivot(index='param_n_neighbors',
columns='param_weights',
values='mean_test_score')



plt.figure(figsize=(10, 12))

sns.heatmap(pivot_table, annot=True,
cmap='RdYlGn', fmt=".4f")

plt.title(f'Hyperparameter Tuning Heatmap
(Metric="{best_metric}")', fontsize=15)

plt.ylabel('K Neighbors')

plt.xlabel('Weighting Strategy')

plt.savefig('tuning_heatmap.png')



# Final Console Output

print("\n--- TUNED MATHEMATICAL RESULTS ---")

print(f"Best Configuration: {best_params}")
```

```
print(f"Accuracy : {tuned_acc:.2f}%")
print(f"Precision: {tuned_prec:.2f}%")
print(f"Recall   : {tuned_rec:.2f}%")
print(f"F1 Score : {tuned_f1:.2f}%")
```

Output:

```
--- TUNED MATHEMATICAL RESULTS ---
Best Configuration: {'metric': 'manhattan', 'n_neighbors': np.int64(9), 'weights': 'uniform'}
Accuracy : 94.00%
Precision: 87.80%
Recall   : 97.30%
F1 Score : 92.31%
```





The optimization process utilized a Grid Search Cross-Validation (5-fold) strategy to navigate a search space consisting of neighbours (k in [1, 30]), distance metrics (L1, L2, and Minkowski), and weighting strategies (uniform vs. distance). This rigorous mathematical approach aimed to move beyond the heuristic baseline of k=5 to identify the global minimum for the model's error rate.

The mathematical transition from the baseline to the tuned configuration—specifically k=9 with Manhattan distance and uniform weights—resulted in a more robust predictive model. Accuracy improved to 94.00%, but the most significant gain was observed in Recall (97.30% vs. 91.89% baseline), demonstrating a vastly superior ability to identify potential purchasers. The F1-score rose to 92.31%, confirming that the trade-off between precision and recall was mathematically optimized.

Technical Justification for Hyperparameter Adjustments

- The increase in neighbours from **5 to 9** was implemented to reduce model Variance. While a lower k is highly sensitive to local outliers, k=9 provides a smoother decision boundary by averaging a larger local neighbourhood, which is visually evidenced in the reduction of "jagged" classification islands.
- The switch to Manhattan Distance (L1 norm) improved stability by calculating absolute differences (d(x, y) = sum |xi - yi|), which is less sensitive to extreme outliers in the

standardized features of Age and Salary compared to the squared terms of the Euclidean metric.

- Finally, Uniform Weights were maintained over distance-based weighting because the dataset density is high enough that an equal majority vote among the 9 closest neighbours provides more reliable generalization than allowing a single proximal point to dominate the classification.

Visual Interpretation

The Local Averaging Analysis (Error Curve) identified k=9 as the "Elbow Point," where the balance between bias and variance is achieved. This is further validated by the Tuning Heatmap, which shows a high-accuracy performance plateau in the [7, 13] neighbour range. The final Tuned Decision Boundary illustrates a refined geometric transition between classes, representing a model that has successfully learned the underlying data distribution while filtering out stochastic noise.

Conclusion:

The implementation of K-Nearest Neighbours (KNN) effectively categorizes behavioural data by leveraging spatial proximity to capture non-linear relationships between variables like Age and Salary. Performance evaluation demonstrates that the model's reliability depends on its ability to filter stochastic noise through a structured neighbourhood, ensuring classification is based on authentic data clusters rather than isolated anomalies. Ultimately, assessing the model through localized decision boundaries proves that KNN is a robust classifier for identifying consumer patterns, achieving high predictive success when the geometric influence of neighbours aligns with the underlying distribution of the feature space.