**Informe del Proyecto: Análisis del Terreno Agrícola “Santiago De Cuba”**

**1. Problemática:**

La agricultura es una actividad fundamental para la economía y la seguridad alimentaria. Sin embargo, la gestión eficiente de los terrenos agrícolas puede ser un desafío debido a la variabilidad en la fertilidad del suelo, los insumos necesarios y la producción esperada. La problemática principal es identificar qué terrenos son más adecuados para ciertos cultivos, optimizando así el uso de recursos y maximizando la producción.

**2. Modelo Computacional Utilizado:**

Para abordar esta problemática, se ha utilizado un modelo de clustering no supervisado, específicamente el algoritmo K-Means. Este algoritmo agrupa los datos en clusters basados en características similares, en este caso, los insumos necesarios y la producción esperada.

**3. Importancia del Modelo:**

El uso de un modelo de clustering como K-Means es crucial en este contexto porque permite:

- Identificar patrones ocultos en los datos sin necesidad de etiquetas predefinidas.

- Agrupar terrenos similares, facilitando la toma de decisiones sobre qué cultivos plantar en cada terreno.

- Optimizar recursos al recomendar terrenos que requieren insumos similares y tienen producciones esperadas comparables.

**4. Pasos Utilizados para Crear el Modelo:**

1- Recopilación de Datos:

- Se creó una base de datos con información sobre diferentes parcelas, incluyendo el tipo de cultivo, los insumos necesarios, la producción esperada y la fertilidad del suelo.

2- Preprocesamiento de Datos:

- Los datos se organizaron en un DataFrame utilizando la biblioteca “pandas”.

3- Selección de Características:

- Se seleccionaron las características relevantes para el clustering: `insumos` y `producción`.

4- Aplicación del Algoritmo K-Means:

- Se aplicó el algoritmo K-Means con 3 clúster para agrupar los terrenos según sus características.

**5. Asignación de Clusters:**

- Cada terreno se asignó a un clúster basado en los resultados del algoritmo K-Means.

**6. Desarrollo de la Interfaz Gráfica:**

- Se creó una interfaz gráfica utilizando “tkinter” para permitir a los usuarios seleccionar un cultivo y obtener recomendaciones sobre los terrenos más adecuados.

**7. Funcionalidad de Recomendación:**

- Se implementaron funciones para obtener recomendaciones basadas en el clúster al que pertenece el cultivo seleccionado.

Análisis del Código con K-Means y Tkinter

**Introducción:**

Este informe describe el funcionamiento del código, el cual utiliza el algoritmo K-Means para agrupar parcelas agrícolas según sus características y proporciona una interfaz gráfica para visualizar las recomendaciones. El código está escrito en Python y utiliza las bibliotecas pandas, scikit-learn y tkinter.

**1. Importación de Bibliotecas:**

import pandas as pd

from sklearn.cluster import KMeans

import tkinter as tk

from tkinter import ttk

pandas: Para la manipulación y análisis de datos.

KMeans: Algoritmo de clustering de scikit-learn.

tkinter y ttk: Para crear la interfaz gráfica.

**2. Configuración del Estilo de la Interfaz:**

def configurar\_estilo():

estilo = ttk.Style()

estilo.theme\_use('clam') # Tema 'clam' para un mejor aspecto

estilo.configure('TLabel', background='#E8E8E8', font=('Helvetica', 10))

estilo.configure('TButton', font=('Helvetica', 10))

estilo.configure('TCombobox', font=('Helvetica', 10))

estilo.map('TCombobox', fieldbackground=[('readonly', '#FFFFFF')])

estilo.map('TCombobox', selectbackground=[('readonly', '#E8E8E8')])

estilo.map('TCombobox', selectforeground=[('readonly', 'black')])

Configura el estilo visual de los widgets de la interfaz gráfica.

**3. Creación de la Base de Datos:**

data = {

'terreno': ['Parcela 1', 'Parcela 2', 'Parcela 3', 'Parcela 4', 'Parcela 5', 'Parcela 6', 'Parcela 7', 'Parcela 8', 'Parcela 9', 'Parcela 10'],

'cultivo': ['Tomate', 'Pepino', 'Arroz', 'Papa', 'Maíz', 'Zanahoria', 'Lechuga', 'Cebolla', 'Frijol', 'Trigo'],

'insumos': [100, 120, 60, 120, 50, 180, 170, 20, 195, 25], # Insumos necesarios

'produccion': [200, 150, 300, 180, 250, 160, 140, 190, 210, 230], # Producción esperada

'infertilidad': [0, 0, 1, 0, 1, 0, 0, 1, 0, 1] # 0: fértil, 1: infértil

}

df = pd.DataFrame(data)

Crea un diccionario con los datos de las parcelas y lo convierte en un DataFrame de pandas.

**4. Aplicación del Algoritmo K-Means:**

¿Qué es K-Means?

K-Means es un algoritmo de clustering que agrupa datos en (k) clusters (grupos) basándose en sus características. El objetivo es minimizar la variación dentro de cada clúster y maximizar la variación entre clusters.

Pasos del Algoritmo:

Inicialización: Selecciona (k) centroides iniciales (puntos centrales de los clusters).

Asignación de Clusters: Asigna cada punto de datos al clúster cuyo centroide esté más cercano.

Actualización de Centroides: Calcula los nuevos centroides como el promedio de los puntos asignados a cada clúster.

Repetición: Repite los pasos 2 y 3 hasta que los centroides no cambien significativamente o se alcance un número máximo de iteraciones.

Aplicación en el Código:

Seleccionar características para clustering:

X = df[['insumos', 'produccion']]

# Aplicar K-Means

kmeans = KMeans(n\_clusters=3, random\_state=42)

df['cluster'] = kmeans.fit\_predict(X)

Selección de Características:

X = df[['insumos', 'produccion']]

Se seleccionan las columnas insumos y produccion del DataFrame df como las características para el clustering.

Inicialización del Modelo K-Means:

kmeans = KMeans(n\_clusters=3, random\_state=42)

Se crea una instancia del modelo K-Means con 3 clusters (n\_clusters=3) y una semilla aleatoria (random\_state=42) para reproducibilidad.

Ajuste y Predicción:

df['cluster'] = kmeans.fit\_predict(X)

El modelo se ajusta a los datos (fit) y se predicen los clusters (predict). Los resultados se almacenan en una nueva columna clúster del DataFrame df.

**5. Funciones para Recomendaciones y Análisis:**

**Función obtener\_recomendaciones:**

def obtener\_recomendaciones(cultivo):

cluster = df[df['cultivo'] == cultivo]['cluster'].values[0]

terrenos\_cluster = df[df['cluster'] == cluster]

return terrenos\_cluster[['terreno', 'insumos', 'produccion', 'infertilidad']]

Propósito: Devuelve las parcelas del mismo clúster que el cultivo seleccionado.

Funcionamiento:

1. Encuentra el clúster al que pertenece el cultivo seleccionado.
2. Filtra las parcelas que pertenecen a ese clúster.
3. Devuelve las columnas terreno, insumos, produccion e infertilidad de las parcelas filtradas.

**Función encontrar\_mejor\_parcela:**

def encontrar\_mejor\_parcela(cultivo):

cluster = df[df['cultivo'] == cultivo]['cluster'].values[0]

terrenos\_cluster = df[df['cluster'] == cluster]

mejor\_parcela = terrenos\_cluster.loc[terrenos\_cluster['infertilidad'] == 0].sort\_values(by=['produccion'], ascending=False).head(1)

return mejor\_parcela['terreno'].values[0] if not mejor\_parcela.empty else 'No hay recomendaciones disponibles'

Propósito: Encuentra la mejor parcela fértil en el mismo clúster que el cultivo seleccionado, basada en la producción.

Funcionamiento:

1. Identificación del Clúster del Cultivo:

cluster = df[df['cultivo'] == cultivo]['cluster'].values[0]

1. Filtra el DataFrame df para encontrar la fila donde el cultivo es igual al cultivo seleccionado.
2. Obtiene el valor del clúster correspondiente a ese cultivo.

Filtrado de Parcelas del Mismo Clúster:

terrenos\_cluster = df[df['cluster'] == cluster]

Filtra el DataFrame df para obtener todas las parcelas que pertenecen al mismo clúster que el cultivo seleccionado.

Filtrado de Parcelas Fértiles:

mejor\_parcela = terrenos\_cluster.loc[terrenos\_cluster['infertilidad'] == 0].sort\_values(by=['produccion'], ascending=False).head(1)

1. Filtra las parcelas del clúster para obtener solo aquellas que son fértiles (infertilidad = 0).
2. Ordena las parcelas fértiles por producción en orden descendente.
3. Selecciona la primera parcela de la lista ordenada, que es la parcela con mayor producción.

Retorno de la Mejor Parcela:

return mejor\_parcela['terreno'].values[0] if not mejor\_parcela.empty else 'No hay recomendaciones disponibles'

1. Verifica si la parcela seleccionada no está vacía.
2. Si no está vacía, devuelve el nombre de la misma.

**Función encontrar\_mejor\_grupo:**

def encontrar\_mejor\_grupo():

grupo\_mejor\_productividad = df.groupby('cluster')['produccion'].mean().idxmax()

return grupo\_mejor\_productividad

Propósito: Encuentra el grupo con la mayor productividad promedio.

Funcionamiento:

1. Agrupación por Clúster: Agrupa las parcelas por la columna clúster.
2. Cálculo de la Productividad Promedio: Calcula la producción promedio de cada clúster.
3. Identificación del Mejor Grupo: Encuentra el índice del clúster con la mayor producción promedio utilizando idxmax().
4. Retorno del Mejor Grupo: Devuelve el índice del clúster con la mayor productividad promedio.

**Función mejor\_parcela\_en\_mejor\_grupo:**

def mejor\_parcela\_en\_mejor\_grupo():

mejor\_grupo = encontrar\_mejor\_grupo()

terrenos\_mejor\_grupo = df[df['cluster'] == mejor\_grupo]

mejor\_parcela = terrenos\_mejor\_grupo.loc[terrenos\_mejor\_grupo['infertilidad'] == 0].sort\_values(by=['produccion'], ascending=False).head(1)

return mejor\_parcela['terreno'].values[0] if not mejor\_parcela.empty else 'No hay recomendaciones disponibles'

Propósito: Encuentra la mejor parcela fértil en el grupo con la mayor productividad promedio.

Funcionamiento:

1. Identificación del Mejor Grupo: Utiliza la función encontrar\_mejor\_grupo() para obtener el índice del clúster con la mayor productividad promedio.
2. Filtrado de Parcelas del Mejor Grupo: Filtra las parcelas que pertenecen a ese clúster.
3. Filtrado de Parcelas Fértiles: Filtra las parcelas fértiles (infertilidad = 0).
4. Ordenación por Producción: Ordena las parcelas fértiles por producción en orden descendente.
5. Selección de la Mejor Parcela: Selecciona la parcela con mayor producción.
6. Retorno de la Mejor Parcela: Devuelve el nombre de la mejor parcela o un mensaje si no hay recomendaciones disponibles.

**Función mejor\_parcela\_para\_cada\_producto:**

def mejor\_parcela\_para\_cada\_producto():

resultados = {}

for cultivo in df['cultivo'].unique():

mejor\_parcela = encontrar\_mejor\_parcela(cultivo)

resultados[cultivo] = mejor\_parcela

return resultados

Propósito: Devuelve la mejor parcela para cada cultivo.

Funcionamiento:

1. Inicialización del Diccionario de Resultados: Crea un diccionario vacío para almacenar los resultados.
2. Iteración sobre Cada Cultivo: Itera sobre cada cultivo único en el DataFrame.
3. Identificación de la Mejor Parcela: Utiliza la función encontrar\_mejor\_parcela() para encontrar la mejor parcela para cada cultivo.
4. Almacenamiento de Resultados: Almacena el nombre de la mejor parcela en el diccionario de resultados.
5. Retorno de Resultados: Devuelve el diccionario con las mejores parcelas para cada cultivo.