PLAN DE TRABAJO DEL ESTUDIANTE

1. INFORMACIÓN GENERAL

Apellidos y Nombres:	Gandy William Humiri Quispe	ID:	1546329@senati.pe
Dirección Zonal/CFP:	Tacna/moquegua		
Carrera:	Ingenieria de Software con Inteligencia Artificial	Semestre:	4ciclo
Curso/ Mód. Formativo	MACHINE LEARNING Y DEEP LEARNING		
Tema del Trabajo:	**Presentar informe primera parte del proyecto final**		

1 . Tipos de algoritmos de aprendizaje supervisado

Algoritmos de aprendizaje supervisado son aquellos que aprenden a partir de datos etiquetados. En este tipo de aprendizaje, el modelo se entrena con un conjunto de datos de entrada donde cada ejemplo de entrenamiento incluye una entrada y una salida deseada. El modelo intenta aprender la relación entre las entradas y las salidas para hacer predicciones sobre nuevos datos.

algoritmos de aprendizaje supervisad

- **Regresión Lineal** se utiliza para predecir valores continuos. por ejemplo, predecir el precio de una casa en función de sus características.
- **K-Nearest Neighbors** Clasifica un punto nuevo en función de las clases de sus vecinos más cercanos en el espacio de características
- **Redes Neuronales** modelos inspirados en el cerebro humano, ideales para problemas complejos como reconocimiento de imágenes o procesamiento del lenguaje natural

```python
# Regresión Lineal Simple
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

tamaño = np.array([[1500], [1600], [1700], [1800], [1900], [2000], [2100], [2200]])
precio = np.array([300000, 320000, 340000, 360000, 380000, 400000, 420000, 440000])

X_train, X_test, y_train, y_test = train_test_split(tamaño, precio, test_size=0.2, random_state=42)

model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
error = mean_squared_error(y_test, y_pred)
print(f"Error cuadrático medio: {error}")
```

```
Error cuadrático medio: 1.6940658945086007e-21
```

2. Regresión Lineal Simple y Múltiple

- **Regresión Lineal Simple:** Es un modelo de regresión que predice un valor continuo utilizando una única variable independiente

  Ejuemplo.

  y=mx+by = mx + by=mx+b

  donde mmm es la pendiente y bbb es el intercepto. Este modelo se usa en situaciones donde solo hay un predictor que afecta el valor de salida

- **Regresión Lineal Múltiple:** Extiende el modelo de regresión lineal simple para trabajar con múltiples variables independientes. La ecuación general es

  y=b0+b1x1+b2x2+⋯+bnxny = b_0 + b_1x_1 + b_2x_2 + \dots + b_nx_ny=b0+b1 x1+b2x2+⋯+bnxn

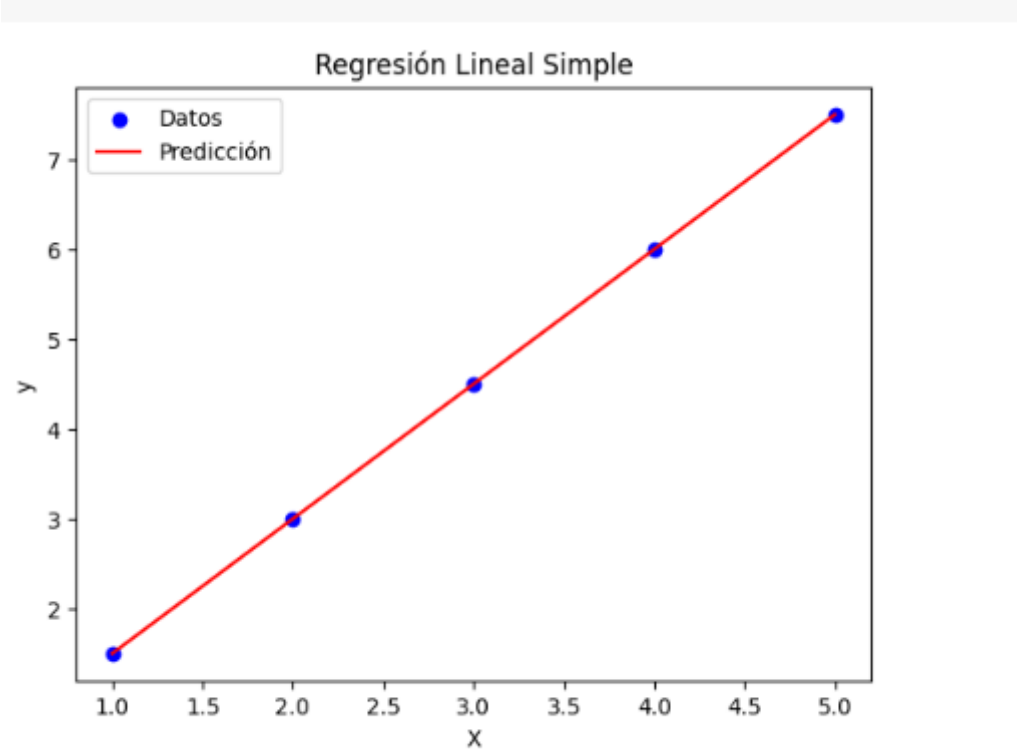
  donde x1,x2,…,xnx_1, x_2, \ldots, x_nx1,x2,…,xn son las variables independientes y b0,b1,…,bnb_0, b_1, \dots, b_nb0,b1,…,bn son los coeficientes que el modelo ajusta para minimizar el error Este modelo se aplica cuando varios factores influyen en la variable objetivo

—

```python
#Clasificación con K-Nearest Neighbors (K-NN)
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

digits = load_digits()
X, y = digits.data, digits.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = KNeighborsClassifier(n_neighbors=5)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Precisión del modelo: {accuracy * 100:.2f}%")
```

```
Precisión del modelo: 98.61%
```

3. Implementación de la Regresión Lineal Simple con Python




```python
import numpy as np
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
X = np.array([[1], [2], [3], [4], [5]])  #  indepññendiente
y = np.array([1.5, 3.0, 4.5, 6.0, 7.5])  # dependiente

model = LinearRegression()
model.fit(X, y)
y_pred = model.predict(X)
plt.scatter(X, y, color='blue', label='Datos')
plt.plot(X, y_pred, color='red', label='Predicción')
plt.xlabel('X')
plt.ylabel('y')
plt.legend()
plt.title('Regresión Lineal Simple')
plt.show()
```

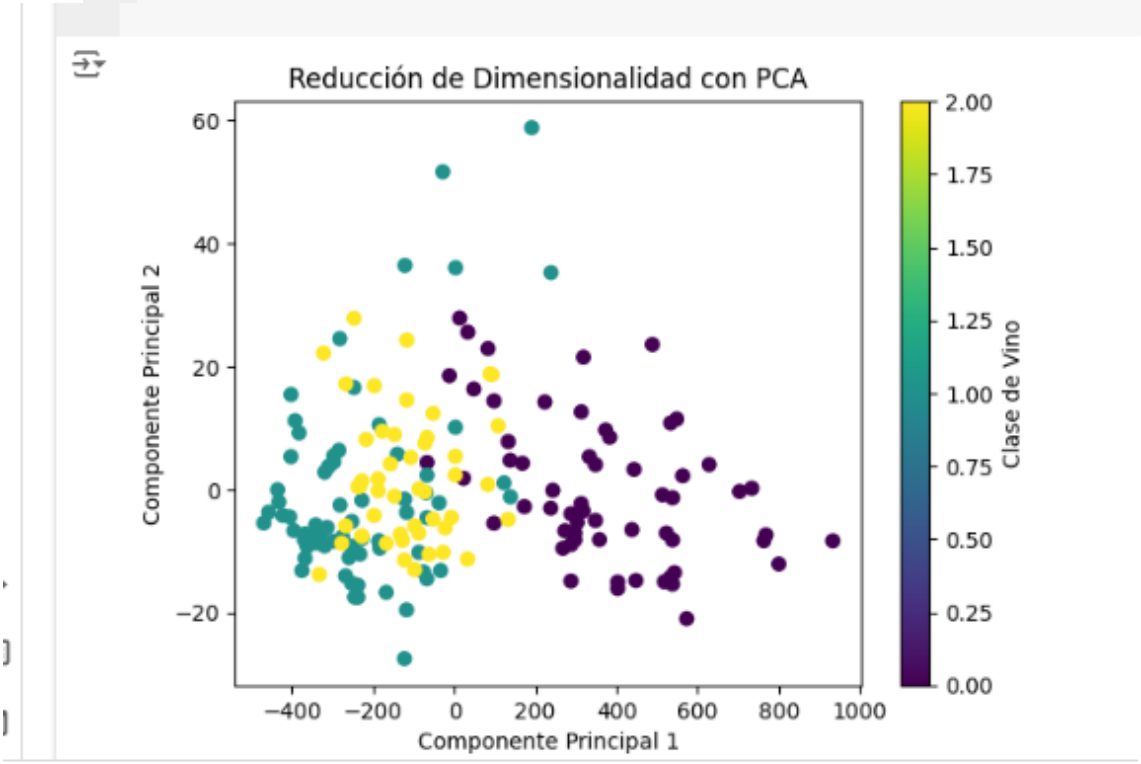
4- Tipos de algoritmos de aprendizaje no supervisado

Algoritmos de aprendizaje no supervisado son aquellos que trabajan con datos no etiquetados y el objetivo es descubrir patrones, agrupaciones o relaciones en los datos sin una salida deseada específica.

algoritmos de aprendizaje no supervisado

- **K-Means:** Algoritmo de agrupamiento que particiona los datos en kkk grupos o clusters
- **Análisis de Componentes Principales :** Reduce la dimensionalidad de los datos, manteniendo las características más relevantes
- **Algoritmo de Agrupamiento Jerárquico:** Forma clusters de datos en una estructura de árbol jerárquica
- **Aprendizaje de Reglas de Asociación :** Busca asociaciones entre elementos, útil en aplicaciones como análisis de canastas de mercado

```python
#Reducción de Dimensionalidad con PCA
from sklearn.datasets import load_wine
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

wine = load_wine()
X = wine.data
y = wine.target

pca = PCA(n_components=2)
X_reducido = pca.fit_transform(X)

plt.scatter(X_reducido[:, 0], X_reducido[:, 1], c=y, cmap='viridis')
plt.xlabel('Componente Principal 1')
plt.ylabel('Componente Principal 2')
plt.colorbar(label='Clase de Vino')
plt.title('Reducción de Dimensionalidad con PCA')
plt.show()
```



5. Diferencias entre algoritmos de clasificación y agrupamiento

- **Clasificación:** Es una técnica supervisada donde los datos ya están etiquetados, y el objetivo es asignare una clase o categoería específicaa a una nueva observación. Ejemplos incluyen SVM y K-NN
- **Agrupamiento (Clustering):** Es una técnica no supervisada donde el objetivo es dividir los datos en grupos o clusters de elementos similares, sin etiquetas predefinidas. Un ejemplo es el algoritmo K-Mean

6. Implementación del algoritmo K-Means con Python

```python
import numpy as np
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
X = np.array([
    [1, 2], [1, 4], [1, 0],
    [10, 2], [10, 4], [10, 0]
])

# Configurar y entrenar el modelo K-Means
kmeans = KMeans(n_clusters=2, random_state=0)
kmeans.fit(X)

labels = kmeans.labels_
centroids = kmeans.cluster_centers_
for i in range(len(labels)):
    plt.scatter(X[i][0], X[i][1], color='blue' if labels[i] == 0 else 'green')

plt.scatter(centroids[:, 0], centroids[:, 1], color='red', marker='X', label='Centroides')
plt.xlabel('X1')
plt.ylabel('X2')
plt.legend()
plt.title('K-Means Clustering')
plt.show()
```