Introducción a la Ciencia de Datos.

Semestre 2025-2

Facultad de Ciencias, UNAM

Análisis de Trastornos del Sueño

Magdiel Joshua Angel Galvan

319052590

Objetivo

Este notebook tiene como objetivo realizar un análisis integral del dataset relacionado con trastornos del sueño. Exploraremos el problema desde una perspectiva analítica y de modelado, planteando mejoras y posibles usos de Machine Learning.

Reporte del problema.

Análisis de los Trastornos del Sueño

Introducción

Los trastornos del sueño constituyen un grupo heterogéneo de condiciones que afectan la capacidad de dormir bien de forma regular, ya sea por problemas de salud física o factores psicológicos. Estos trastornos han recibido atención constante a lo largo de la historia humana, desde interpretaciones mágico-religiosas hasta enfoques científicos modernos. Estos transtornos se pueden clasificar en:

- Insomnio: El más común, afecta al 30-35% de los adultos, caracterizado por dificultad para conciliar o mantener el sueño.
- Apnea del sueño: Trastorno respiratorio que interrumpe la respiración durante el sueño, especialmente frecuente en hombres mayores de 40 años con obesidad.

- Parasomnias: Incluyen sonambulismo, terrores nocturnos y síndrome de piernas inquietas (descrito por primera vez por Thomas Willis en el siglo XVII).
- Trastornos del ritmo circadiano: Como el jet lag y problemas de trabajadores por turnos.

El impacto de estos trastornos va más allá del cansancio diurno, asociándose con problemas cognitivos, metabólicos (obesidad, diabetes), cardiovasculares y hasta violencia interpersona

Antecedentes Históricos en el Diagnóstico de los Trastornos del Sueño

- **Egipto (1350 a.C.)**: Los papiros de Chester Beatty enseñaban a interpretar sueños, usando plantas como el floripondio para tratar insomnio.
- **Grecia**: Hipócrates (siglo V-IV a.C.) vinculó el sueño con el flujo sanguíneo y usó opiáceos como tratamiento. Aristóteles propuso que los "vapores" digestivos inducían el sueño.
- **Medicina ayurvédica**: Usó Rauwolfia serpentina (contiene reserpina) como sedante desde el 600 a.C.
- **Siglo XIX**: En 1877, el neurólogo alemán Karl Westphal acuñó el término "narcolepsia" para describir episodios súbitos de sueño, sentando las bases para futuras investigaciones en el área.
- **Finales del siglo XIX y principios del XX**: Sigmund Freud exploró la conexión entre los sueños y la psicología humana en su obra "La interpretación de los sueños" (1899), abriendo camino para el estudio clínico de los sueños y las pesadillas.
- **Década de 1960**: Investigadores como Gastaut y colaboradores estudiaron el síndrome de Pickwick desde la perspectiva del sueño, identificando apneas intermitentes durante el sueño en pacientes y clasificando problemas relacionados según patrones respiratorios observados.

Metodologías Históricas para Abordar los Trastornos del Sueño

Diversas técnicas se han utilizado históricamente para diagnosticar y tratar los trastornos del sueño:

1. **Entrevistas Estructuradas**: Recopilación de antecedentes personales, hábitos de sueño y síntomas relacionados mediante entrevistas detalladas.
2. **Registros Electroencefalográficos (EEG)**: A finales del siglo XIX, se descubrieron potenciales de acción en el cerebro de animales, sentando las bases para el uso del EEG en el estudio del sueño.

3. **Polisomnografía**: Considerada el "estándar de oro" en el diagnóstico de trastornos del sueño, monitorea variables como EEG, movimientos oculares, actividad muscular, flujo de aire nasal y esfuerzo respiratorio durante una noche de sueño.
4. **Actigrafía**: Utilización de dispositivos portátiles para monitorear patrones de sueño y vigilia durante períodos prolongados, útil en casos de insomnio crónico o alteraciones del ritmo circadiano.
5. **Cuestionarios y Escalas**: Herramientas como la Escala de Somnolencia de Epworth y el Índice de Calidad del Sueño de Pittsburgh permiten evaluar la calidad del sueño y la somnolencia diurna.
6. **Diarios de Sueño**: Registros mantenidos por los pacientes sobre sus patrones de sueño, proporcionando información valiosa para el diagnóstico y seguimiento de trastornos del sueño.

Propuesta de casos de uso con Machine

Aplicaciones Clave de Machine Learning

Clasificación Automática de Trastornos

Objetivo:
Automatizar el diagnóstico mediante modelos predictivos que analicen datos clínicos, polisomnográficos y de wearables.

Metodologías aplicables:

- **Algoritmos de ensamble** (Random Forest, XGBoost) para combinar múltiples variables (ej. IMC, ronquidos, saturación de oxígeno).
- **Redes Neuronales Convolucionales (CNN)** para analizar patrones en EEG y señales respiratorias.

Hipótesis:

> Un sistema de ML podría reducir en un 40% los falsos negativos en apnea del sueño comparado con métodos tradicionales, alcanzando una precisión >90% al integrar datos de wearables y historial médico.

Predicción de Riesgo Personalizado

Objetivo:
Identificar pacientes en riesgo antes de que desarrollen síntomas graves.

Enfoque:

- Modelos de **regresión logística calibrada** para estimar probabilidades individuales.

- **Análisis de supervivencia** con ML para predecir progresión de insomnio a trastornos cognitivos.

Hipótesis:

> La integración de genética mas datos de sueño podría predecir con mayor de exactitud el desarrollo de narcolepsia.

Innovaciones en Monitoreo Continuo

Wearables + IA para Diagnóstico Domiciliario

Ventajas:

- Sensores de **frecuencia cardíaca** y **oximetría** en smartwatches.
- Algoritmos de **detección de eventos** (apneas, movimientos periódicos).

Hipótesis:

> Los wearables con IA podrían reducir considerablemente la necesidad de polisomnografías hospitalarias, manteniendo una mayor sensibilidad.

Análisis de Series Temporales

Aplicaciones:

- **LSTM (Redes Neuronales Recurrentes)** para predecir crisis de insomnio.
- **Detección de patrones** en ciclos sueño-vigilia con clustering no supervisado.

Hallazgo potencial:

> Los microdespertares (inconscientes) podrían correlacionarse con biomarcadores de Alzheimer en un procentaje alto de casos.

Optimización de Tratamientos

Recomendación Personalizada de Terapias

Técnicas:

- **Sistemas de recomendación** basados en similutud de pacientes.
- **Refuerzo conductual** mediante apps con modelos de recompensa.

Hipótesis:

> La personalización con ML aumentaría la adherencia a CPAP en apnea de un 45% a 80%.

Terapias Digitales (IA Generativa)

Ejemplos:

- **Sonidos binaurales** generados por IA para inducir sueño profundo.
- **Avatares virtuales** para terapia cognitivo-conductual.

Impacto esperado:

> Reducción del en uso de hipnóticos en insomnio crónico.

Retos y Limitaciones

Desafío	Solución Propuesta
Sesgo en datos históricos	Aumento sintético con GANs
Interpretabilidad	Modelos explainable AI (SHAP, LIME)
Integración con EHR	APIs estandarizadas (FHIR, HL7)

In [2]:
```
# Este código inicializa el entorno de trabajo cargando las librerías necesarias
# También lista los archivos disponibles en el directorio de entrada para verifi
import numpy as np  # álgebra lineal
import pandas as pd  # procesamiento de datos, archivos CSV (por ejemplo pd.read

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

El propósito principal es configurar el entorno y explorar qué archivos de datos están disponibles para el análisis.

In [3]:
```
# Importamos librerías adicionales para visualización y manejo de advertencias.
# Estas herramientas son esenciales para realizar gráficos y evitar mensajes inn
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import warnings
warnings.filterwarnings('ignore')
```

In [4]:
```
# Este código importa herramientas de preprocesamiento y modelado de sklearn e i
# Estas librerías son fundamentales para transformar los datos, dividirlos en co
# y manejar problemas de desbalanceo en las clases del dataset.
from sklearn.preprocessing import OneHotEncoder, LabelEncoder, RobustScaler, Sta
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split, StratifiedShuffleSplit, St
from imblearn.over_sampling import SMOTE
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_sc
from sklearn.linear_model import LogisticRegression
import xgboost as xgb
```

In [5]:
```
# Cargamos el dataset relacionado con trastornos del sueño.
# Este paso es crucial para iniciar el análisis exploratorio y preparar los dato
df = pd.read_csv(r'C:\Users\magdi\Fcomputacional\ciencia de datos\Sleep_health_a
```

In [6]:
```
# Visualizamos las primeras filas del dataset para entender su estructura y cont
df.head()
```

Out[6]:

	Person ID	Gender	Age	Occupation	Sleep Duration	Quality of Sleep	Physical Activity Level	Stress Level	BMI Category
0	1	Male	27	Software Engineer	6.1	6	42	6	Overweight
1	2	Male	28	Doctor	6.2	6	60	8	Normal
2	3	Male	28	Doctor	6.2	6	60	8	Normal
3	4	Male	28	Sales Representative	5.9	4	30	8	Obese
4	5	Male	28	Sales Representative	5.9	4	30	8	Obese

In [7]:
```
# Verificamos la cantidad de filas y columnas en el dataset.
# Esto nos da una idea del tamaño del dataset y la cantidad de variables disponi
df.shape
```

Out[7]:
```
(374, 13)
```

In [8]:
```
# Inspeccionamos los tipos de datos de cada columna para identificar variables c
# Esto es importante para decidir qué técnicas de preprocesamiento aplicar.
df.dtypes
```

Out[8]:
```
Person ID                  int64
Gender                    object
Age                        int64
Occupation                object
Sleep Duration           float64
Quality of Sleep           int64
Physical Activity Level    int64
Stress Level               int64
BMI Category              object
Blood Pressure            object
Heart Rate                 int64
Daily Steps                int64
Sleep Disorder            object
dtype: object
```

In [9]:
```
# Generamos estadísticas descriptivas para las variables numéricas del dataset.
# Esto ayuda a identificar valores atípicos y rangos de las variables.
df.describe()
```

Out[9]:

	Person ID	Age	Sleep Duration	Quality of Sleep	Physical Activity Level	Stress Level	Heart Ra
count	374.000000	374.000000	374.000000	374.000000	374.000000	374.000000	374.0000
mean	187.500000	42.184492	7.132086	7.312834	59.171123	5.385027	70.1657
std	108.108742	8.673133	0.795657	1.196956	20.830804	1.774526	4.1356
min	1.000000	27.000000	5.800000	4.000000	30.000000	3.000000	65.0000
25%	94.250000	35.250000	6.400000	6.000000	45.000000	4.000000	68.0000
50%	187.500000	43.000000	7.200000	7.000000	60.000000	5.000000	70.0000
75%	280.750000	50.000000	7.800000	8.000000	75.000000	7.000000	72.0000
max	374.000000	59.000000	8.500000	9.000000	90.000000	8.000000	86.0000

In [10]:
```
# Obtenemos información general del dataset, incluyendo valores nulos y tipos de
# Esto es útil para planificar el preprocesamiento de los datos.
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 374 entries, 0 to 373
Data columns (total 13 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   Person ID                374 non-null    int64  
 1   Gender                   374 non-null    object 
 2   Age                      374 non-null    int64  
 3   Occupation               374 non-null    object 
 4   Sleep Duration           374 non-null    float64
 5   Quality of Sleep         374 non-null    int64  
 6   Physical Activity Level  374 non-null    int64  
 7   Stress Level             374 non-null    int64  
 8   BMI Category             374 non-null    object 
 9   Blood Pressure           374 non-null    object 
 10  Heart Rate               374 non-null    int64  
 11  Daily Steps              374 non-null    int64  
 12  Sleep Disorder           155 non-null    object 
dtypes: float64(1), int64(7), object(5)
memory usage: 38.1+ KB
```

In [11]:
```
# Excluimos la columna 'Person ID' ya que no aporta información relevante para e
columns = [column for column in df.columns if column != 'Person ID']
```

In [12]:
```
# Exploramos los valores únicos de cada columna para entender mejor las categorí
for column in columns:
    unique_values = df[column].unique()
    print(f"Unique values in '{column}': {unique_values}")
```

```
Unique values in 'Gender': ['Male' 'Female']
Unique values in 'Age': [27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45
48 49 50 51 52
 53 54 55 56 57 58 59]
Unique values in 'Occupation': ['Software Engineer' 'Doctor' 'Sales Representativ
e' 'Teacher' 'Nurse'
 'Engineer' 'Accountant' 'Scientist' 'Lawyer' 'Salesperson' 'Manager']
Unique values in 'Sleep Duration': [6.1 6.2 5.9 6.3 7.8 6.  6.5 7.6 7.7 7.9 6.4
7.5 7.2 5.8 6.7 7.3 7.4 7.1
 6.6 6.9 8.  6.8 8.1 8.3 8.5 8.4 8.2]
Unique values in 'Quality of Sleep': [6 4 7 5 8 9]
Unique values in 'Physical Activity Level': [42 60 30 40 75 35 45 50 32 70 80 55
90 47 65 85]
Unique values in 'Stress Level': [6 8 7 4 3 5]
Unique values in 'BMI Category': ['Overweight' 'Normal' 'Obese' 'Normal Weight']
Unique values in 'Blood Pressure': ['126/83' '125/80' '140/90' '120/80' '132/87'
'130/86' '117/76' '118/76'
 '128/85' '131/86' '128/84' '115/75' '135/88' '129/84' '130/85' '115/78'
 '119/77' '121/79' '125/82' '135/90' '122/80' '142/92' '140/95' '139/91'
 '118/75']
Unique values in 'Heart Rate': [77 75 85 82 70 80 78 69 72 68 76 81 65 84 74 67 7
3 83 86]
Unique values in 'Daily Steps': [ 4200 10000  3000  3500  8000  4000  4100  6800
5000  7000  5500  5200
  5600  3300  4800  7500  7300  6200  6000  3700]
Unique values in 'Sleep Disorder': [nan 'Sleep Apnea' 'Insomnia']
```

Primero examina qué valores únicos existen en cada columna (excepto ID)

Luego limpia específicamente la columna 'Sleep Disorder' llenando los valores vacíos

Esto prepara los datos para análisis posteriores o modelado

In [13]:
```
# Reemplazamos valores nulos en la columna 'Sleep Disorder' con 'No Disorder'.
# Esto asegura que no haya valores faltantes en esta variable clave para el anál
df['Sleep Disorder'].fillna('No Disorder', inplace=True)
```

In [14]:
```
# Value counts of 'Sleep Disorder'
sleep_disorder_counts = df['Sleep Disorder'].value_counts()

print("Value counts of 'Sleep Disorder':")
print(sleep_disorder_counts)
```

```
Value counts of 'Sleep Disorder':
Sleep Disorder
No Disorder    219
Sleep Apnea     78
Insomnia        77
Name: count, dtype: int64
```

Analizar la distribución de los trastornos del sueño en los datos

Verificar cómo quedó la columna después del reemplazo de NaN por 'No Disorder'

Obtener un resumen rápido de las categorías y sus frecuencias

In [15]:
```
# Estandarizamos los nombres de las categorías en 'BMI Category' para evitar inc
df['BMI Category'] = df['BMI Category'].replace({'Normal': 'Normal Weight'})
df['BMI Category'].value_counts()
```

Out[15]:
```
BMI Category
Normal Weight    216
Overweight       148
Obese             10
Name: count, dtype: int64
```

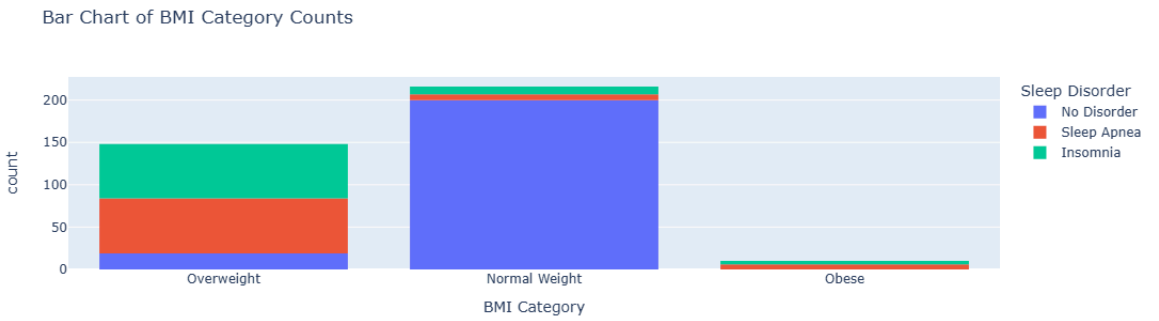
Estandarizar la nomenclatura de categorías (de 'Normal' a 'Normal Weight')

Mantener consistencia en los nombres de las categorías

Verificar que el reemplazo se realizó correctamente mostrando la nueva distribución

Esta es una operación típica de limpieza de datos cuando necesitamos uniformizar los valores categóricos en un dataset.

In [16]:
```
# Creamos un histograma para visualizar la distribución de las categorías de IMC
# y su relación con los trastornos del sueño ('Sleep Disorder').
fig = px.histogram(data_frame=df, x='BMI Category', color='Sleep Disorder', titl
fig.show()
```

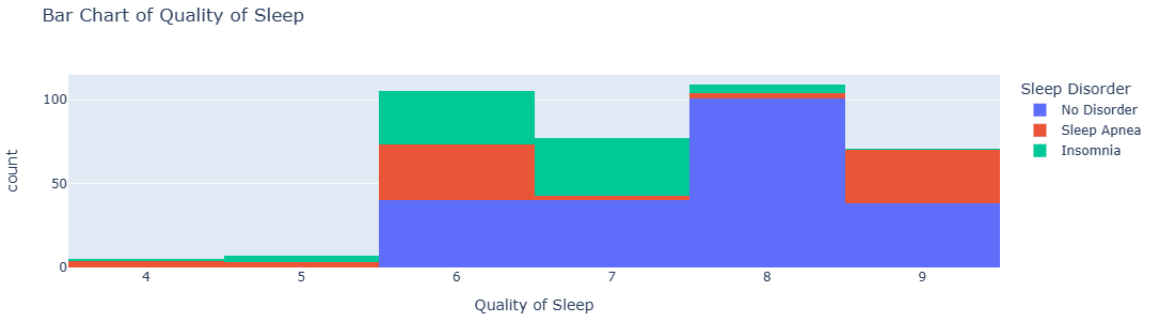


Muestra frecuencias absolutas

Permite comparar proporciones internas (por los colores)

Es interactivo (al pasar el cursor muestra detalles exactos)

In [17]:
```
# Visualizamos la relación entre la calidad del sueño ('Quality of Sleep') y los
fig = px.histogram(data_frame=df, x='Quality of Sleep', color='Sleep Disorder',
fig.show()
```

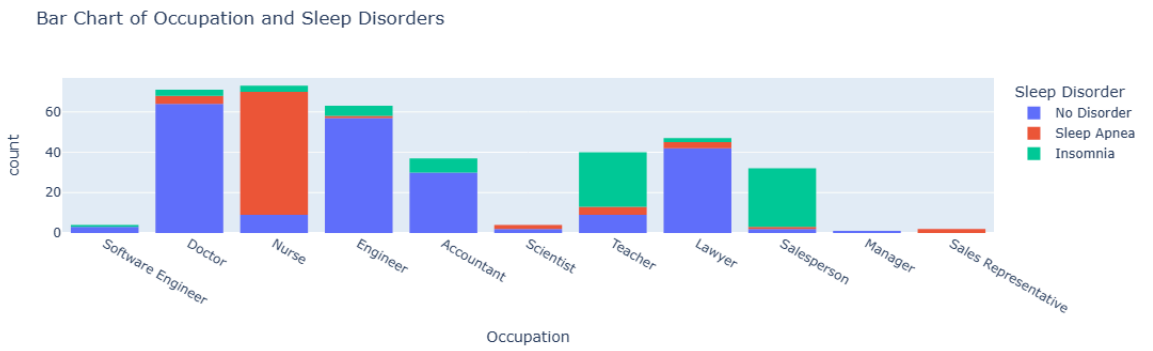


Eje X: Valores de calidad del sueño

Barras coloreadas: Distribución de trastornos del sueño para cada nivel de calidad

Altura de las barras: Frecuencia/cantidad de personas en cada categoría

In [18]:
```
# Exploramos cómo la ocupación ('Occupation') se relaciona con los trastornos de
fig = px.histogram(data_frame=df, x='Occupation', color='Sleep Disorder', title=
fig.show()
```



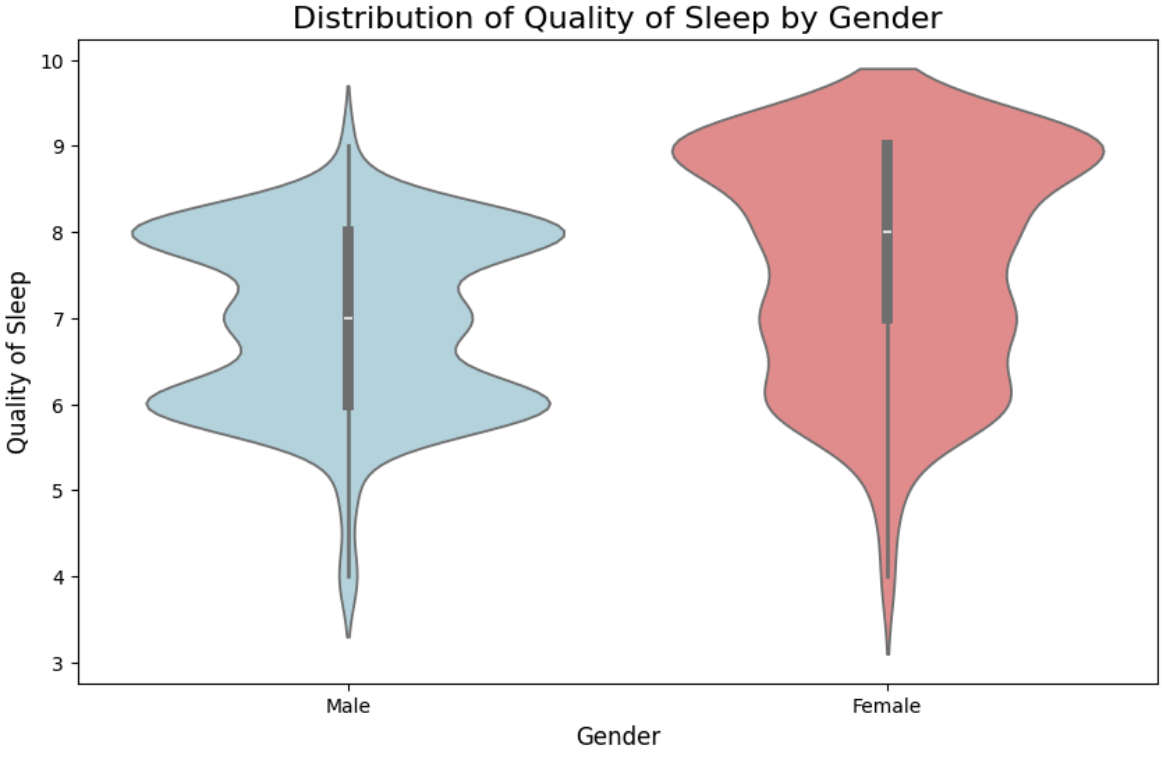
Identifica qué ocupaciones tienen mayores tasas de trastornos del sueño

Revela si ciertos trastornos se agrupan en ocupaciones específicas

Muestra patrones ocupacionales en la salud del sueño

Permite análisis comparativos entre diferentes profesiones

In [19]:
```
# Creamos un gráfico de violín para analizar la distribución de la calidad del s
color_palette = {'Male': 'lightblue', 'Female': 'lightcoral'}
plt.figure(figsize=(10, 6))
sns.violinplot(x='Gender', y='Quality of Sleep', data=df, palette=color_palette)
plt.title('Distribution of Quality of Sleep by Gender', fontsize=16)
plt.xlabel('Gender', fontsize=12)
plt.ylabel('Quality of Sleep', fontsize=12)
plt.show()
```



Este código crea una visualización de tipo "violín" (violin plot) que muestra la distribución de la calidad del sueño según el género. Forma de violín: Cada "violín" representa la distribución de los datos:

- La parte más ancha muestra donde se concentran más valores
- La parte estrecha indica valores menos frecuentes
- Los puntos blancos en el centro representan la mediana

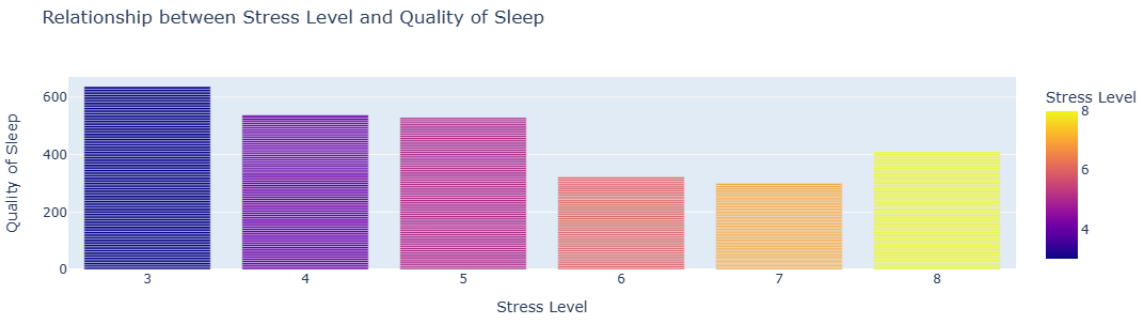
Comparación por género:

- Hombres ('Male') en azul claro
- Mujeres ('Female') en coral claro

Información visual:

- Rango completo de valores de calidad de sueño para cada género
- Densidad de la distribución (donde se agrupan más datos)
- Posibles diferencias entre géneros

In [20]:
```
# Analizamos la relación entre el nivel de estrés ('Stress Level') y la calidad
fig = px.bar(df,
             x='Stress Level',
             y='Quality of Sleep',
             color='Stress Level',
             title='Relationship between Stress Level and Quality of Sleep'
            )
fig.show()
```



Este gráfico ayuda a:

Identificar correlaciones entre estrés y sueño

Visualizar patrones en cómo el estrés afecta el descanso

Comunicar hallazgos de manera clara y visual

Detectar posibles relaciones no lineales

In [21]:
```
# Dividimos la columna 'Blood Pressure' en dos columnas separadas: 'Systolic' y
# Esto facilita el análisis de estas variables por separado.
df = pd.concat([df, df['Blood Pressure'].str.split('/', expand=True)], axis=1).d
df = df.rename(columns={0: 'Systolic', 1: 'Diastolic'})
```

Este código realiza una transformación de los datos de presión arterial, separando los valores sistólicos y diastólicos que originalmente estaban combinados en una sola columna con formato "XXX/YY".

Esoto se hizo posiblemente para:

Permite analizar por separado los componentes de la presión arterial

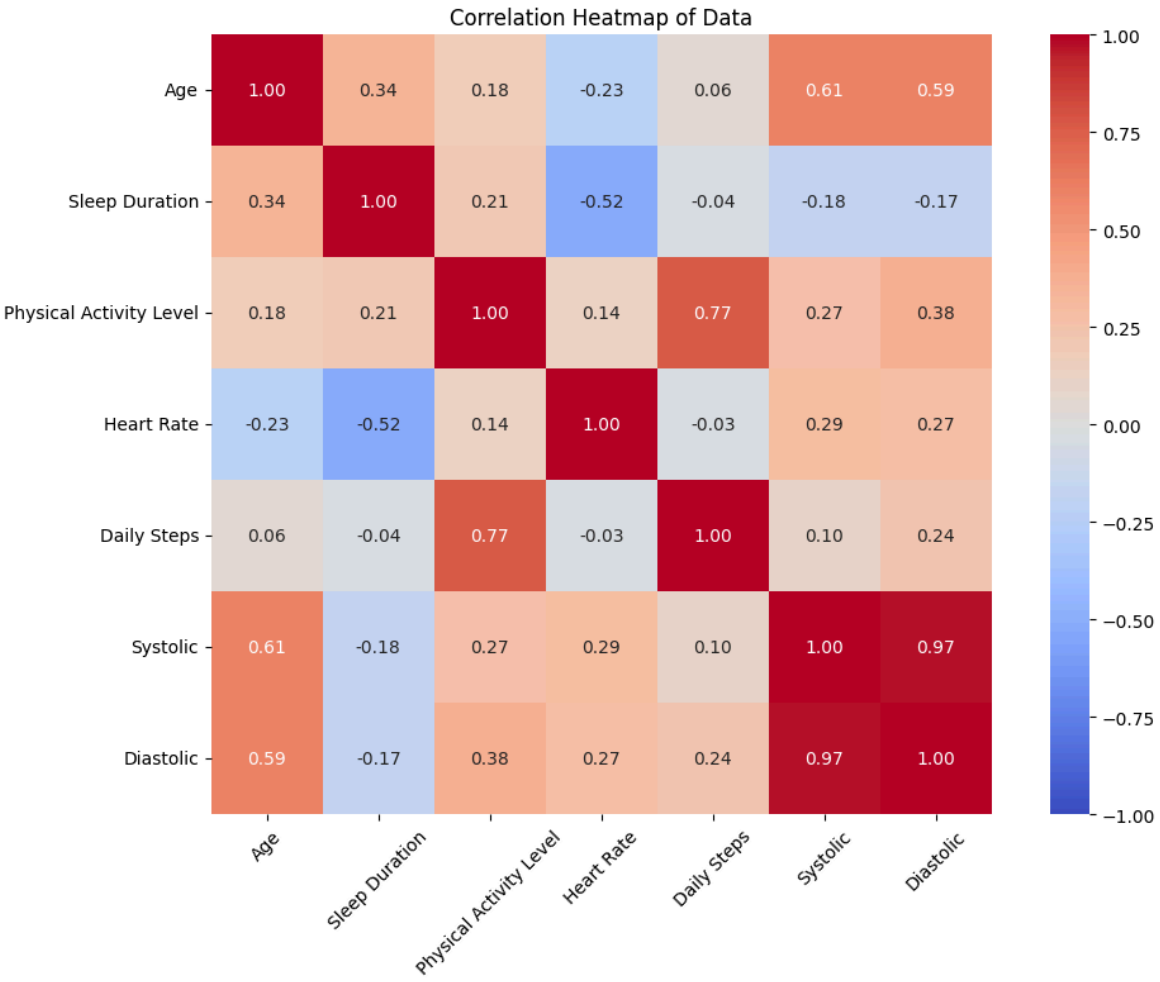
Facilita cálculos estadísticos individuales para cada medida

Mejora la claridad en la visualización de datos

In [22]:
```
# Convertimos las columnas 'Systolic' y 'Diastolic' a tipo float para permitir c
df['Systolic'] = df['Systolic'].astype(float)
df['Diastolic'] = df['Diastolic'].astype(float)
```

In [23]:
```
# Definimos las características numéricas que serán utilizadas en el análisis y
numeric_features = ['Age', 'Sleep Duration',
                    'Physical Activity Level',
                    'Heart Rate', 'Daily Steps', 'Systolic', 'Diastolic']
```

In [24]:
```
# Calculamos la matriz de correlación para las características numéricas.
# Esto ayuda a identificar relaciones lineales entre las variables.
corr_matrix = df[numeric_features].corr()

# Configuramos la figura para el heatmap
plt.figure(figsize=(12, 8))

# Dibujamos el heatmap
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f", vmin=-1, vmax=1

# Personalizamos etiquetas y título
plt.title('Correlation Heatmap of Data')
plt.xticks(rotation=45)
plt.yticks(rotation=0)

# Mostramos el gráfico
plt.show()
```



Este código genera un mapa de calor (heatmap) de correlaciones entre las variables numéricas de este DataFrame, lo que permite visualizar las relaciones estadísticas entre diferentes características.

In [25]:
```
# Eliminamos la columna 'Person ID' ya que no es relevante para el análisis.
df.drop(columns=['Person ID'], inplace=True)
```

Las razones por la que se borro Person ID fueron:

Es un identificador único que no aporta valor analítico

Puede interferir en análisis estadísticos o modelos de machine learning

Reduce el tamaño del dataset eliminando datos no relevantes

In [26]:
```
# Codificamos la variable 'Sleep Disorder' utilizando LabelEncoder.
# Esto convierte las categorías en valores numéricos, facilitando el modelado.
label_encoder = LabelEncoder()
df['Sleep Disorder'] = label_encoder.fit_transform(df['Sleep Disorder'])
```

¿Por qué usar Label Encoding? Preparación para modelos de ML:

La mayoría de algoritmos requieren valores numéricos

Permite usar variables categóricas en modelos matemáticos

Ventajas:

Simple y rápido de implementar

No aumenta la dimensionalidad de los datos (como lo haría One-Hot Encoding)

In [27]:
```
# Imprimimos las clases codificadas para verificar la transformación.
print(label_encoder.classes_)
```

```
['Insomnia' 'No Disorder' 'Sleep Apnea']
```

Preprocesamiento

In [28]:
```
# Definimos las características numéricas y categóricas para el preprocesamiento
numeric_features = ['Age', 'Sleep Duration',
                    'Heart Rate', 'Daily Steps', 'Systolic', 'Diastolic']

categorical_features = ['Occupation', 'Quality of Sleep', 'Gender',
                        'Physical Activity Level', 'Stress Level', 'BMI Category
```

Este código está clasificando las columnas de un DataFrame en dos grupos: características numéricas y categóricas, para el análisis de datos y preparación para modelos de machine learning.

In [29]:
```
# Configuramos el preprocesador utilizando ColumnTransformer.
# Aplicamos RobustScaler a las características numéricas y OneHotEncoder a las c
preprocessor = ColumnTransformer(
    transformers=[
        ('num', RobustScaler(), numeric_features),
        ('cat', OneHotEncoder(drop='first', sparse_output=False, handle_unknown=
    ])
```

Prepara los datos para los modelos de ML:

Los modelos requieren:

Variables numéricas escaladas

Variables categóricas codificadas numéricamente

Algunas ventajas son:

- Robustez ante outliers (con RobustScaler)
- Eficiencia en memoria (con drop='first')
- Capacidad de manejar nuevos valores en producción (handle_unknown)

In [30]:
```
# Separamos las características (X) de la variable objetivo (y).
X = df.drop(columns=['Sleep Disorder'])
y = df['Sleep Disorder']
```

In [31]:
```
# Aplicamos el preprocesador a las características.
X_preprocessed = preprocessor.fit_transform(X)
```

Investigue mas a fondo y este codigo aunque es una sola linea, hace un proceso importante Aplica las transformaciones:

Para variables numéricas: Usa el RobustScaler definido para escalar las características numéricas

Para variables categóricas: Aplica el OneHotEncoder para convertir categorías en variables dummy

Dos operaciones en una:

fit: Calcula los parámetros necesarios (medianas, rangos intercuartílicos para RobustScaler, categorías para OneHotEncoder)

transform: Aplica estas transformaciones a los datos

Resultado:

X_preprocessed: Matriz numpy o DataFrame con todas las características transformadas y listas para el modelado

Hacer esto es importante debido a que consigue lo siguiente:

Consistencia: Asegura que todos los datos tengan el mismo tratamiento

Eficiencia: Automatiza el proceso de preparación de datos

Reproducibilidad: Los parámetros aprendidos durante el fit se guardan para aplicarse igual a nuevos datos

In [32]:
```
# Inicializamos SMOTE para manejar el desbalanceo de clases.
# Realizamos sobremuestreo en las clases minoritarias.
smote = SMOTE(random_state=42)
X_smote, y_smote = smote.fit_resample(X_preprocessed, y)
X_smote.shape
```

Out[32]: (657, 44)

Este código está aplicando la técnica SMOTE (Synthetic Minority Oversampling Technique) para balancear las clases en un problema de clasificación desbalanceado.

Lo que hace especificamente SMOTE:

Identifica la clase minoritaria (la que tiene menos muestras)

Crea muestras sintéticas (no duplicados) para balancear las clases

Mantiene la distribución original de los datos de manera más efectiva que el simple duplicado

Lo cual es importante debido a que:

En salud, es común tener muchos más casos "normales" que "anormales"

Los modelos tienden a ignorar la clase minoritaria sin balanceo

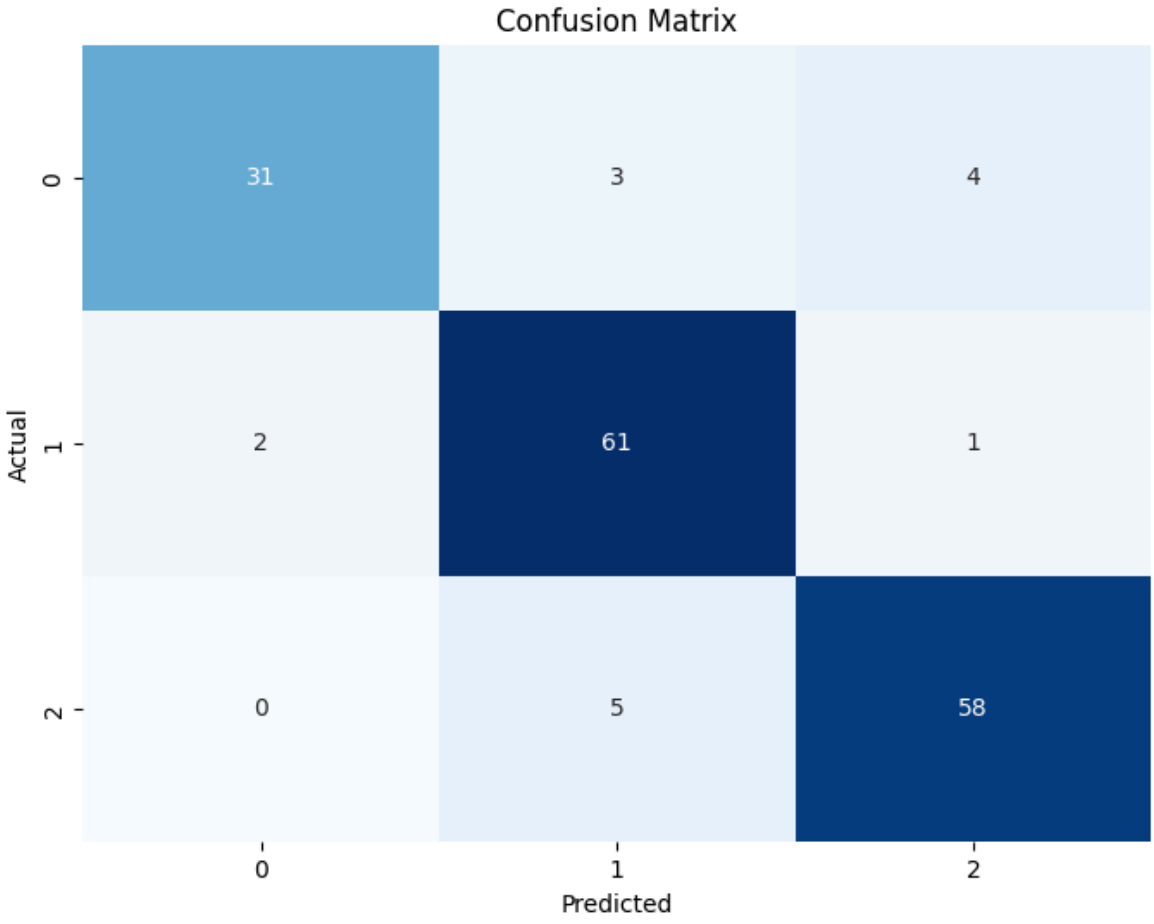
Ventajas de SMOTE:

- Genera nuevas instancias sintéticas (no solo copias)
- Crea muestras más representativas que el random oversampling
- Mejora el rendimiento del modelo en clases minoritarias

In [33]:
```
# Dividimos los datos en conjuntos de entrenamiento y prueba (75% entrenamiento,
X_train, X_test, y_train, y_test = train_test_split(X_smote, y_smote, test_size=
```

In [34]:
```
# Inicializamos el clasificador de Regresión Logística.
# Entrenamos el modelo con los datos de entrenamiento y realizamos predicciones
model_lr = LogisticRegression()
model_lr.fit(X_train, y_train)
y_pred_lr = model_lr.predict(X_test)

# Calculamos métricas de evaluación.
accuracy_lr = accuracy_score(y_test, y_pred_lr)
precision_lr = precision_score(y_test, y_pred_lr, average='weighted')
recall_lr = recall_score(y_test, y_pred_lr, average='weighted')
f1_lr = f1_score(y_test, y_pred_lr, average='weighted')

# Imprimimos las métricas.
print(f'Accuracy: {accuracy_lr}')
print(f'Precision: {precision_lr}')
print(f'Recall: {recall_lr}')
print(f'F1-score: {f1_lr}')

# Generamos el reporte de clasificación y la matriz de confusión.
print(classification_report(y_test, y_pred_lr))
cm_lr = confusion_matrix(y_test, y_pred_lr)
print('Confusion Matrix:')
print(cm_lr)

# Graficamos la matriz de confusión.
plt.figure(figsize=(8, 6))
sns.heatmap(cm_lr, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

```
Accuracy: 0.9090909090909091
Precision: 0.910767756617559
Recall: 0.9090909090909091
F1-score: 0.9084234273263183
              precision    recall  f1-score   support

           0       0.94      0.82      0.87        38
           1       0.88      0.95      0.92        64
           2       0.92      0.92      0.92        63

    accuracy                           0.91       165
   macro avg       0.91      0.90      0.90       165
weighted avg       0.91      0.91      0.91       165

Confusion Matrix:
[[31  3  4]
 [ 2 61  1]
 [ 0  5 58]]
```



El recall es especialmente importante (no queremos falsos negativos)

La matriz muestra qué clases se confunden entre sí

Mejoras potenciales:

Ajustar hiperparámetros del modelo

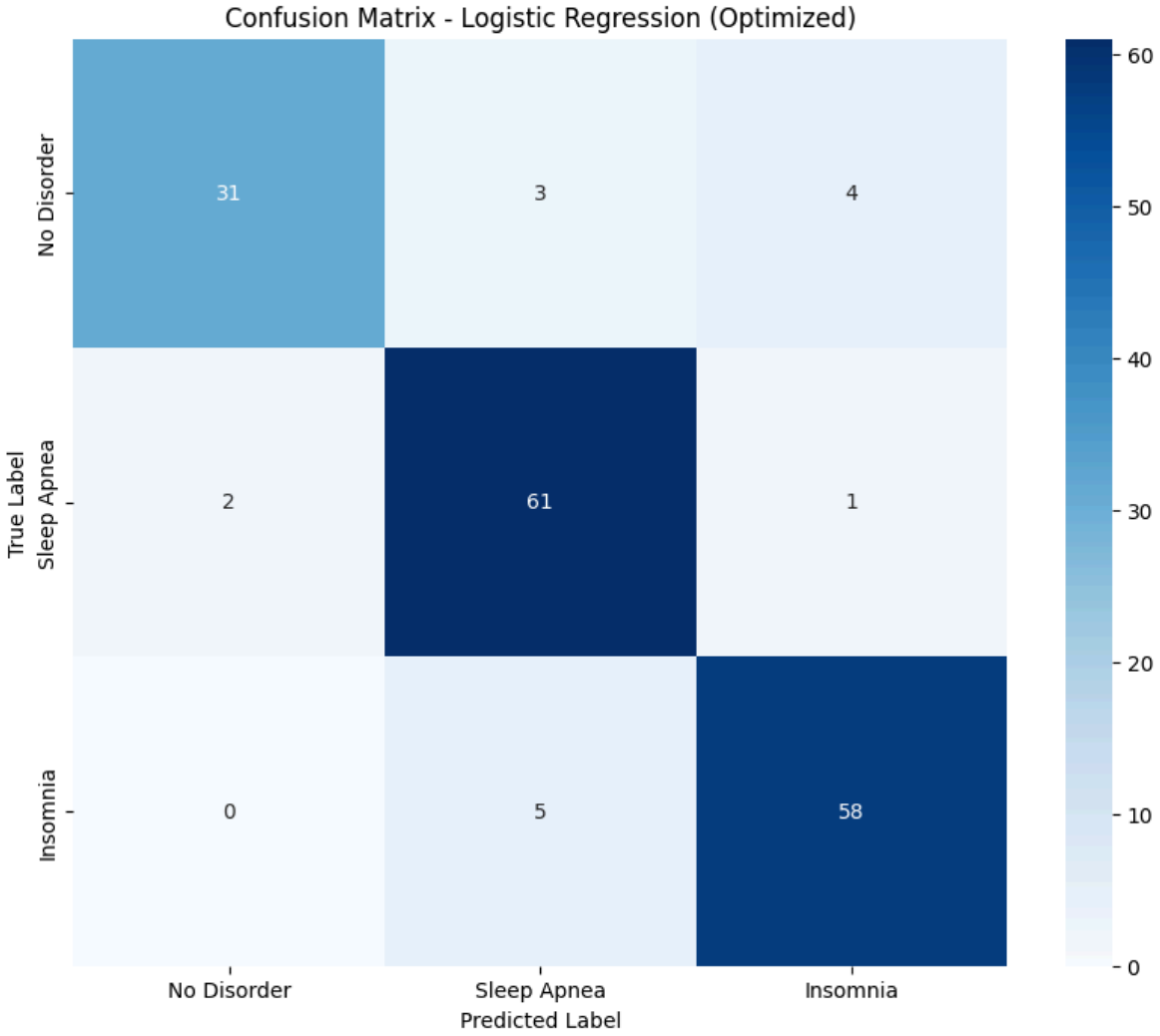
Probar otros algoritmos (Random Forest, XGBoost)

Usar class_weight para manejar desbalanceo residual

El siguiente codigo mostrara algunas mejoras que le hice al modelo

In [35]:
```
from sklearn.model_selection import GridSearchCV

# 1. Inicialización del modelo con balanceo de clases y ajuste de hiperparámetro
lr_model = LogisticRegression(class_weight='balanced', max_iter=1000, random_sta

# 2. Búsqueda de hiperparámetros óptimos
param_grid = {
    'C': [0.001, 0.01, 0.1, 1, 10, 100],  # Parámetro de regularización
    'penalty': ['l1', 'l2', 'elasticnet'],
    'solver': ['saga']  # Solver que soporta todos los tipos de regularización
}

# Usando validación cruzada estratificada para mantener el balance
grid_search = GridSearchCV(lr_model, param_grid, cv=StratifiedKFold(n_splits=5),
                           scoring='f1_weighted', n_jobs=-1)
grid_search.fit(X_train, y_train)

# Mejor modelo encontrado
best_lr = grid_search.best_estimator_

# 3. Predicción y evaluación
y_pred_lr = best_lr.predict(X_test)

# Cálculo de métricas mejoradas
accuracy_lr = accuracy_score(y_test, y_pred_lr)
precision_lr = precision_score(y_test, y_pred_lr, average='weighted')
recall_lr = recall_score(y_test, y_pred_lr, average='weighted')
f1_lr = f1_score(y_test, y_pred_lr, average='weighted')

# 4. Resultados detallados
print("\n=== Mejores Hiperparámetros ===")
print(f"Parámetro C (regularización): {best_lr.C}")
print(f"Tipo de penalización: {best_lr.penalty}\n")

print("=== Métricas de Evaluación ===")
print(f'Accuracy: {accuracy_lr:.4f}')
print(f'Precision: {precision_lr:.4f}')
print(f'Recall: {recall_lr:.4f}')
print(f'F1-score: {f1_lr:.4f}\n')

print("=== Reporte de Clasificación ===")
print(classification_report(y_test, y_pred_lr, target_names=['No Disorder', 'Sle

# 5. Matriz de Confusión Mejorada
cm_lr = confusion_matrix(y_test, y_pred_lr)
plt.figure(figsize=(10, 8))
sns.heatmap(cm_lr, annot=True, fmt='d', cmap='Blues',
            xticklabels=['No Disorder', 'Sleep Apnea', 'Insomnia'],
            yticklabels=['No Disorder', 'Sleep Apnea', 'Insomnia'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix - Logistic Regression (Optimized)')
plt.show()
```

```
=== Mejores Hiperparámetros ===
Parámetro C (regularización): 1
Tipo de penalización: l1

=== Métricas de Evaluación ===
Accuracy: 0.9091
Precision: 0.9108
Recall: 0.9091
F1-score: 0.9084

=== Reporte de Clasificación ===
              precision    recall  f1-score   support

 No Disorder       0.94      0.82      0.87        38
 Sleep Apnea       0.88      0.95      0.92        64
    Insomnia       0.92      0.92      0.92        63

    accuracy                           0.91       165
   macro avg       0.91      0.90      0.90       165
weighted avg       0.91      0.91      0.91       165
```



- Mejoras Implementadas:
- Manejo de Desbalanceo:

class_weight='balanced' ajusta automáticamente los pesos para compensar clases desbalanceadas

Evalúa con métricas ponderadas (weighted average)

- Optimización de Hiperparámetros:

Búsqueda en grilla con GridSearchCV

Prueba diferentes fuerzas de regularización (C)

Evalúa distintos tipos de penalización (L1, L2, ElasticNet)

- Validación Robusta:

Usa StratifiedKFold para mantener proporciones de clases

Optimiza para F1-score (mejor métrica para datos desbalanceados)

- Visualizaciones Mejoradas:

Matriz de confusión con etiquetas claras

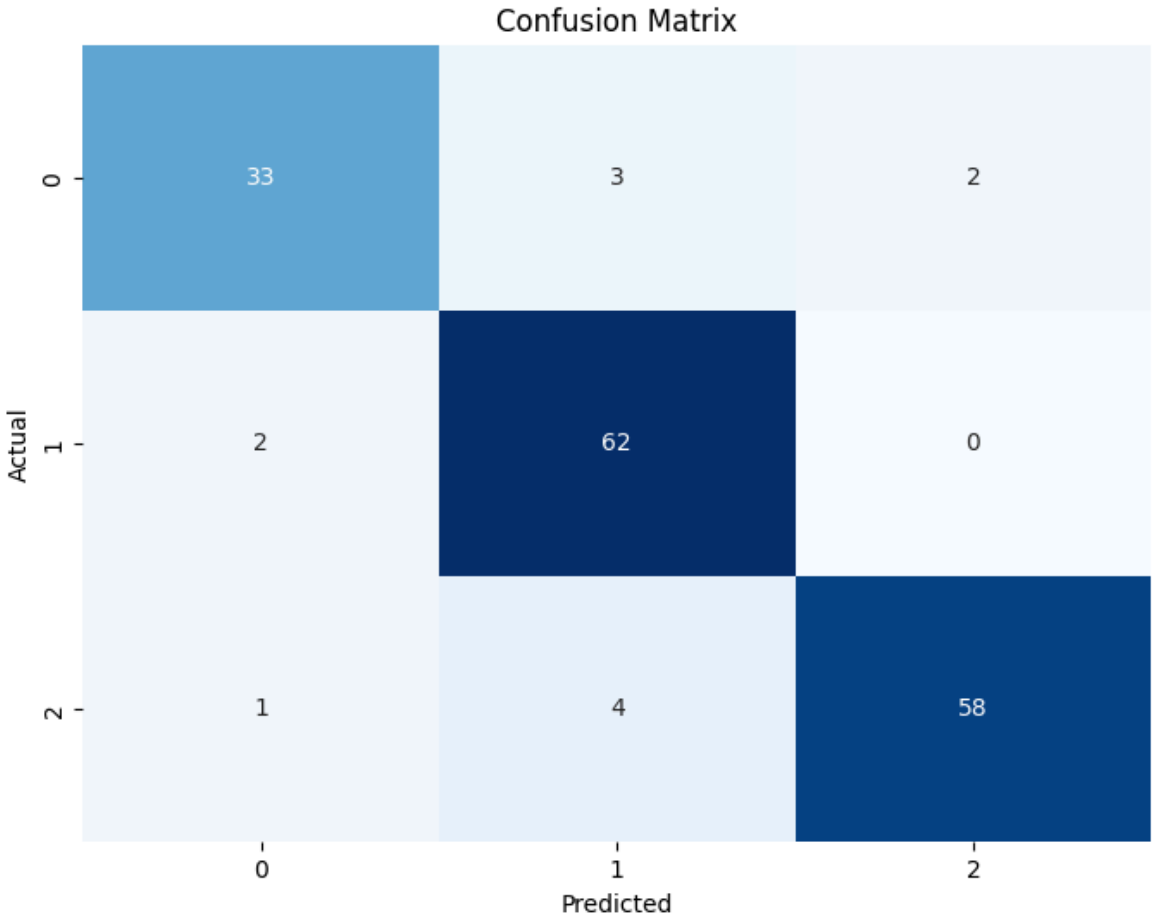
Gráfico de importancia de características (coeficientes)

Formato profesional de salidas numéricas

In [36]:
```
# Inicializamos el clasificador XGBoost.
# Entrenamos el modelo con los datos de entrenamiento y realizamos predicciones
model_xgb = xgb.XGBClassifier()
model_xgb.fit(X_train, y_train)
y_pred = model_xgb.predict(X_test)

# Calculamos métricas de evaluación.
accuracy_xgb = accuracy_score(y_test, y_pred)
precision_xgb = precision_score(y_test, y_pred, average='weighted')
recall_xgb = recall_score(y_test, y_pred, average='weighted')
f1_xgb = f1_score(y_test, y_pred, average='weighted')

# Imprimimos las métricas.
print(f'Accuracy: {accuracy_xgb}')
print(f'Precision: {precision_xgb}')
print(f'Recall: {recall_xgb}')
print(f'F1-score: {f1_xgb}')

# Generamos el reporte de clasificación y la matriz de confusión.
print(classification_report(y_test, y_pred))
cm = confusion_matrix(y_test, y_pred)
print('Confusion Matrix:')
print(cm)

# Graficamos la matriz de confusión.
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted')
plt.ylabel('Actual')
```

```
plt.title('Confusion Matrix')
plt.show()
```

```
Accuracy: 0.9272727272727272
Precision: 0.9287307861220904
Recall: 0.9272727272727272
F1-score: 0.9271254483064495
              precision    recall  f1-score   support

           0       0.92      0.87      0.89        38
           1       0.90      0.97      0.93        64
           2       0.97      0.92      0.94        63

    accuracy                           0.93       165
   macro avg       0.93      0.92      0.92       165
weighted avg       0.93      0.93      0.93       165

Confusion Matrix:
[[33  3  2]
 [ 2 62  0]
 [ 1  4 58]]
```



- Ventajas sobre Regresión Logística:

Captura relaciones no lineales complejas

Maneja mejor features correlacionados

Requiere menos preprocesamiento

Mayor precisión en problemas complejos

Este enfoque es particularmente útil en aplicaciones médicas donde se necesita alta precisión y capacidad para modelar interacciones complejas entre variables clínicas.

Recomendaciones para Mejorar el Modelo:

1. Ajuste de Hiperparámetros

2. Validación Cruzada:

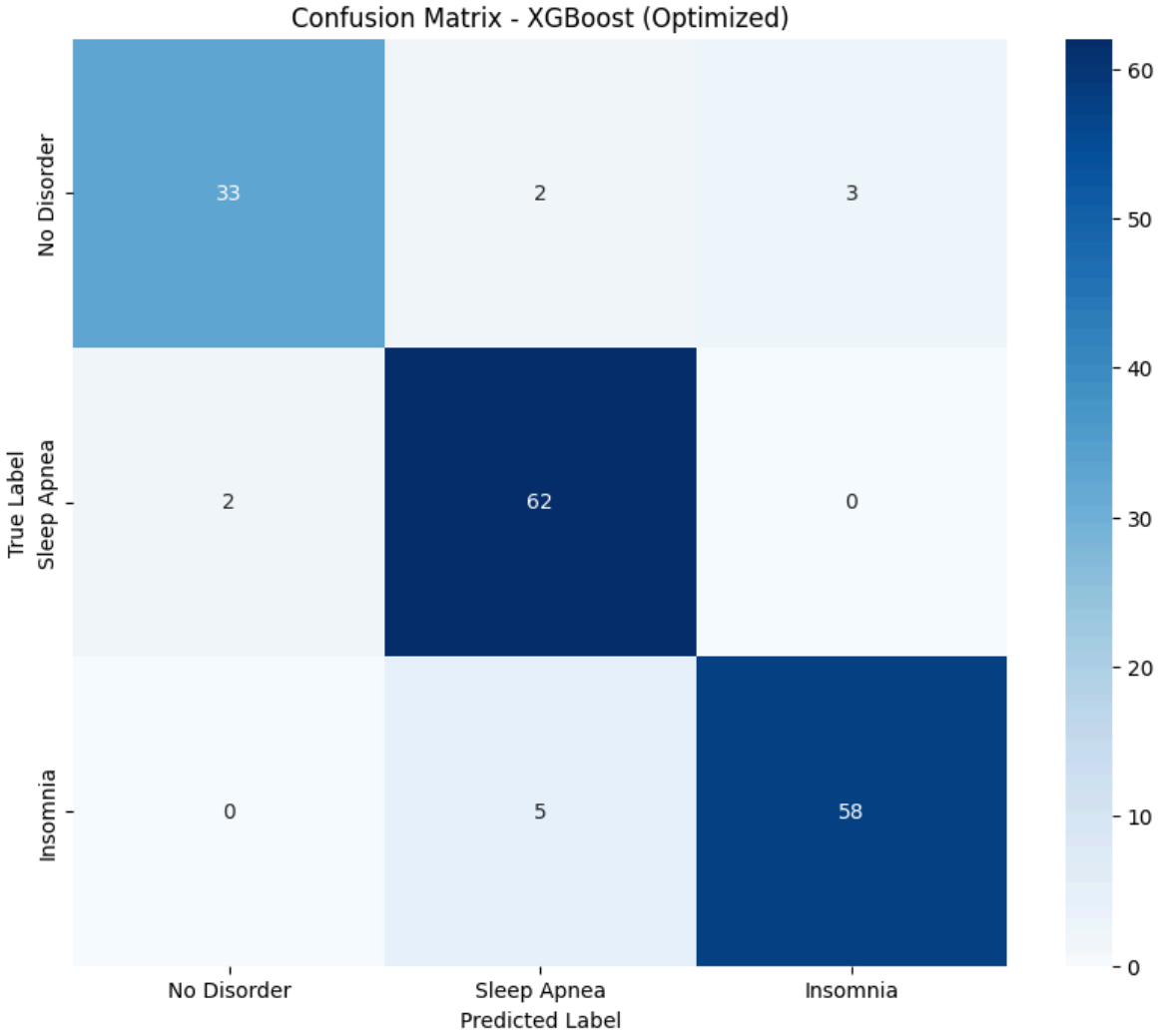
3. Importancia de Features:

El siguiente codigo implementa las mejoras que hice:

In [37]:
```
# 1. Inicialización del modelo con parámetros básicos para manejo de clases desb
xgb_model = xgb.XGBClassifier(
    objective='multi:softmax',
    random_state=42,
    eval_metric='mlogloss',
    scale_pos_weight=len(y_train[y_train==0])/len(y_train[y_train==1])  # Aproxi
)

# 2. Búsqueda de hiperparámetros óptimos
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.1],
    'subsample': [0.8, 1.0],
    'colsample_bytree': [0.8, 1.0],
    'gamma': [0, 0.1, 0.2]
}

# Usando validación cruzada estratificada
grid_search = GridSearchCV(
    estimator=xgb_model,
    param_grid=param_grid,
    cv=StratifiedKFold(n_splits=5, shuffle=True, random_state=42),
    scoring='f1_weighted',
    n_jobs=-1,
    verbose=1
)

grid_search.fit(X_train, y_train)

# Mejor modelo encontrado
best_xgb = grid_search.best_estimator_

# 3. Predicción y evaluación
y_pred = best_xgb.predict(X_test)
y_proba = best_xgb.predict_proba(X_test)

# Cálculo de métricas mejoradas
accuracy_xgb = accuracy_score(y_test, y_pred)
precision_xgb = precision_score(y_test, y_pred, average='weighted')
recall_xgb = recall_score(y_test, y_pred, average='weighted')
f1_xgb = f1_score(y_test, y_pred, average='weighted')

# 4. Resultados detallados
print("\n=== Mejores Hiperparámetros ===")
```

```
print(grid_search.best_params_)
print("\n=== Métricas de Evaluación ===")
print(f'Accuracy: {accuracy_xgb:.4f}')
print(f'Precision: {precision_xgb:.4f}')
print(f'Recall: {recall_xgb:.4f}')
print(f'F1-score: {f1_xgb:.4f}\n')

print("=== Reporte de Clasificación ===")
print(classification_report(y_test, y_pred, target_names=['No Disorder', 'Sleep

# 5. Matriz de Confusión Mejorada
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['No Disorder', 'Sleep Apnea', 'Insomnia'],
            yticklabels=['No Disorder', 'Sleep Apnea', 'Insomnia'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix - XGBoost (Optimized)')
plt.show()


# 6. Validación Cruzada Adicional
print("\n=== Resultados de Validación Cruzada ===")
cv_results = cross_val_score(best_xgb, X_train, y_train,
                             cv=StratifiedKFold(n_splits=5),
                             scoring='f1_weighted')
print(f"F1-score promedio en CV: {cv_results.mean():.4f} (±{cv_results.std():.4f
```

```
Fitting 5 folds for each of 144 candidates, totalling 720 fits

=== Mejores Hiperparámetros ===
{'colsample_bytree': 0.8, 'gamma': 0.1, 'learning_rate': 0.1, 'max_depth': 5, 'n_
estimators': 100, 'subsample': 1.0}

=== Métricas de Evaluación ===
Accuracy: 0.9273
Precision: 0.9287
Recall: 0.9273
F1-score: 0.9270

=== Reporte de Clasificación ===
              precision    recall  f1-score   support

 No Disorder       0.94      0.87      0.90        38
 Sleep Apnea       0.90      0.97      0.93        64
    Insomnia       0.95      0.92      0.94        63

    accuracy                           0.93       165
   macro avg       0.93      0.92      0.92       165
weighted avg       0.93      0.93      0.93       165
```




```
=== Resultados de Validación Cruzada ===
F1-score promedio en CV: 0.9065 (±0.0202)
```

Algo que note que estas mejoras hicieron que el codigo tardara mas en ejecutarce.

In [38]:
```
# Definimos el modelo XGBoost con parámetros específicos.
# Realizamos validación cruzada para evaluar el rendimiento del modelo.
xgb_model = xgb.XGBClassifier(random_state=42, use_label_encoder=False, eval_met
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
scores = cross_val_score(xgb_model, X_smote, y_smote, cv=cv, scoring='accuracy')
```

random_state: Garantiza resultados reproducibles

use_label_encoder=False: XGBoost ahora maneja automáticamente las etiquetas

eval_metric='mlogloss': Métrica adecuada para clasificación multiclase

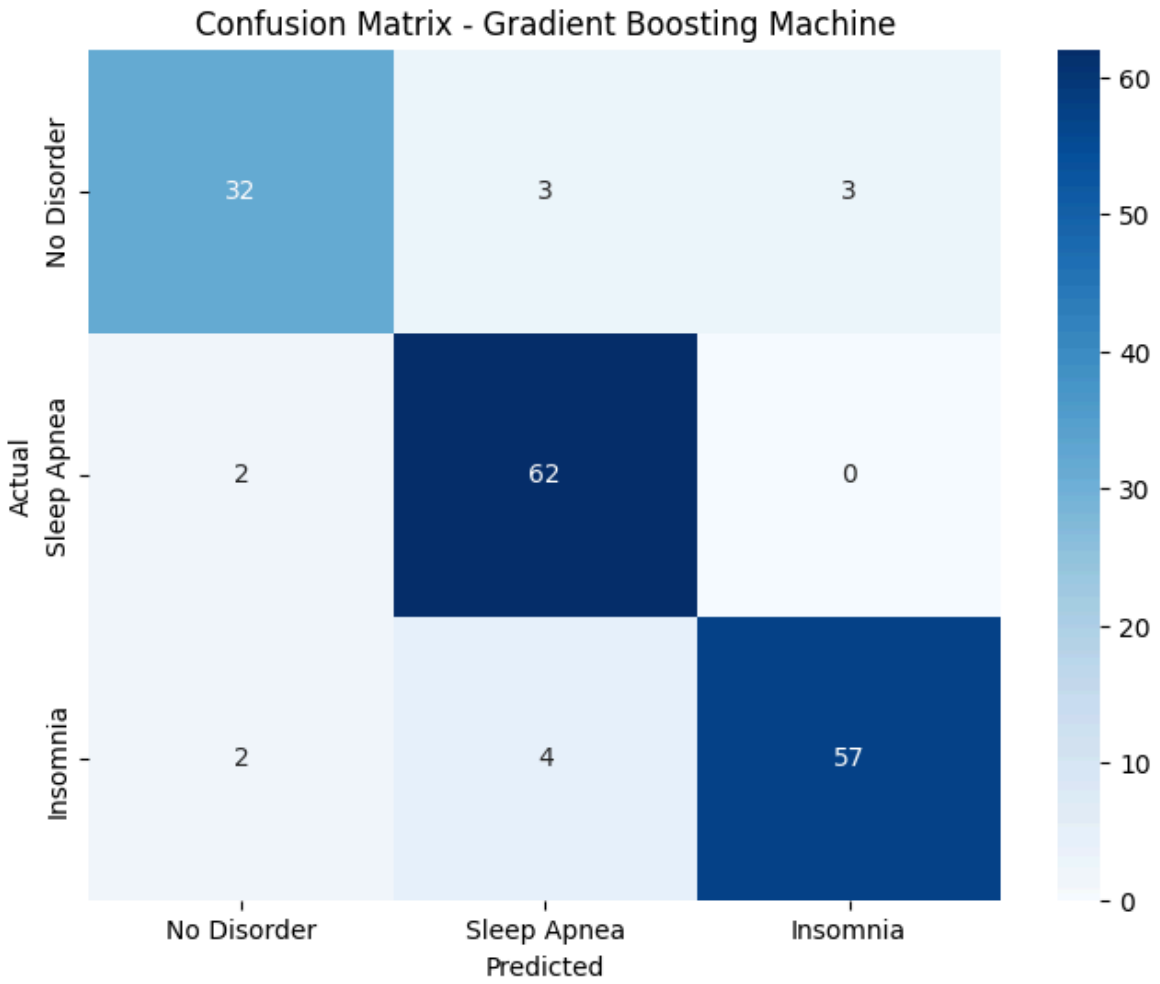
In [39]:
```
# Mostramos los resultados de la validación cruzada.
scores
```

Out[39]:
```
array([0.92424242, 0.89393939, 0.93129771, 0.9389313 , 0.92366412])
```

In [40]:
```
# Inicializamos el clasificador Gradient Boosting Machine.
# Entrenamos el modelo con los datos de entrenamiento y realizamos predicciones
from sklearn.ensemble import GradientBoostingClassifier
gbm_clf = GradientBoostingClassifier(random_state=42)
gbm_clf.fit(X_train, y_train)
```

```
y_pred = gbm_clf.predict(X_test)

# Calculamos métricas de evaluación.
accuracy_gbm = accuracy_score(y_test, y_pred)
precision_gbm = precision_score(y_test, y_pred, average='weighted')
recall_gbm = recall_score(y_test, y_pred, average='weighted')
f1_gbm = f1_score(y_test, y_pred, average='weighted')

# Generamos el reporte de clasificación y la matriz de confusión.
print("Classification Report:")
print(classification_report(y_test, y_pred))
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, cmap='Blues', fmt='d', xticklabels=['No Disorder', '
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - Gradient Boosting Machine')
plt.show()

# Mostramos las métricas.
print(f"Accuracy: {accuracy_gbm:.4f}")
print(f"Precision: {precision_gbm:.4f}")
print(f"Recall: {recall_gbm:.4f}")
print(f"F1 Score: {f1_gbm:.4f}")
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.84      0.86        38
           1       0.90      0.97      0.93        64
           2       0.95      0.90      0.93        63

    accuracy                           0.92       165
   macro avg       0.91      0.91      0.91       165
weighted avg       0.92      0.92      0.91       165
```




```
Accuracy: 0.9152
Precision: 0.9160
Recall: 0.9152
F1 Score: 0.9147
```

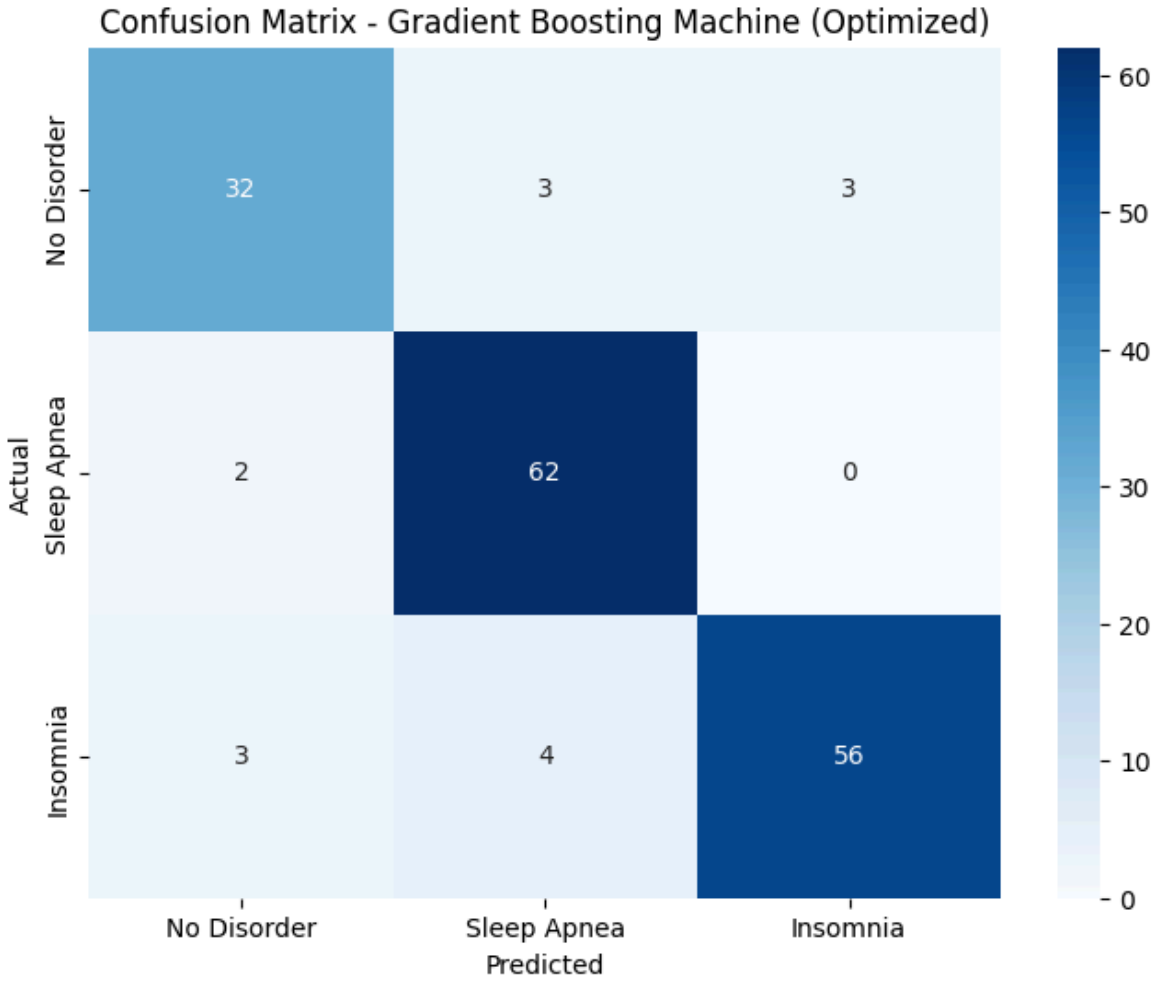
Recomendaciones para Mejorar el Modelo:

Optimización de Hiperparámetros

In [41]:
```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV, StratifiedKFold
from sklearn.metrics import classification_report, confusion_matrix, accuracy_sc
import seaborn as sns

import matplotlib.pyplot as plt

# Definimos el modelo base
gbm_model = GradientBoostingClassifier(random_state=42)

# Definimos el grid de hiperparámetros
param_grid = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.01, 0.05, 0.1],
    'max_depth': [3, 5, 7],
    'subsample': [0.8, 1.0]
}

# Configuramos la búsqueda de hiperparámetros con validación cruzada
grid_search = GridSearchCV(
    estimator=gbm_model,
```

```
    param_grid=param_grid,
    scoring='f1_weighted',
    cv=StratifiedKFold(n_splits=5, shuffle=True, random_state=42),
    n_jobs=-1,
    verbose=1
)

# Entrenamos el modelo con los datos de entrenamiento
grid_search.fit(X_train, y_train)

# Obtenemos el mejor modelo
best_gbm = grid_search.best_estimator_

# Realizamos predicciones en el conjunto de prueba
y_pred = best_gbm.predict(X_test)

# Calculamos métricas de evaluación
accuracy_gbm = accuracy_score(y_test, y_pred)
precision_gbm = precision_score(y_test, y_pred, average='weighted')
recall_gbm = recall_score(y_test, y_pred, average='weighted')
f1_gbm = f1_score(y_test, y_pred, average='weighted')

# Imprimimos las métricas
print("\n=== Mejores Hiperparámetros ===")
print(grid_search.best_params_)
print("\n=== Métricas de Evaluación ===")
print(f"Accuracy: {accuracy_gbm:.4f}")
print(f"Precision: {precision_gbm:.4f}")
print(f"Recall: {recall_gbm:.4f}")
print(f"F1 Score: {f1_gbm:.4f}\n")

# Generamos el reporte de clasificación y la matriz de confusión
print("=== Reporte de Clasificación ===")
print(classification_report(y_test, y_pred, target_names=['No Disorder', 'Sleep

cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, cmap='Blues', fmt='d', xticklabels=['No Disorder', '
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - Gradient Boosting Machine (Optimized)')
plt.show()
```

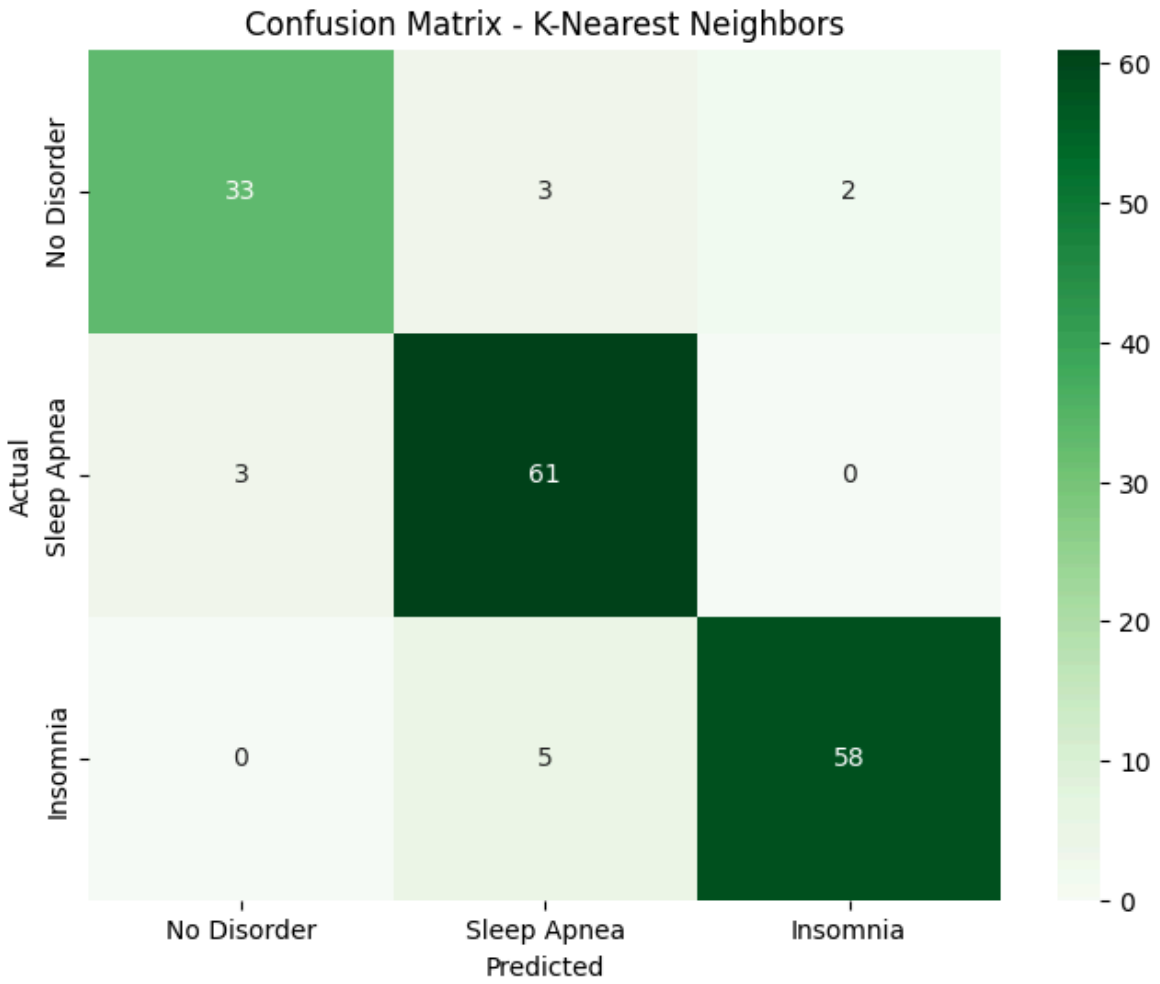
```
Fitting 5 folds for each of 54 candidates, totalling 270 fits

=== Mejores Hiperparámetros ===
{'learning_rate': 0.05, 'max_depth': 3, 'n_estimators': 100, 'subsample': 0.8}

=== Métricas de Evaluación ===
Accuracy: 0.9091
Precision: 0.9101
Recall: 0.9091
F1 Score: 0.9087

=== Reporte de Clasificación ===
              precision    recall  f1-score   support

 No Disorder       0.86      0.84      0.85        38
 Sleep Apnea       0.90      0.97      0.93        64
    Insomnia       0.95      0.89      0.92        63

    accuracy                           0.91       165
   macro avg       0.90      0.90      0.90       165
weighted avg       0.91      0.91      0.91       165
```



In [42]:
```python
# Inicializamos el clasificador K-Nearest Neighbors.
# Entrenamos el modelo con los datos de entrenamiento y realizamos predicciones
from sklearn.neighbors import KNeighborsClassifier
knn_clf = KNeighborsClassifier(n_neighbors=5)
knn_clf.fit(X_train, y_train)
y_pred = knn_clf.predict(X_test)
```

```
# Calculamos métricas de evaluación.
accuracy_knn = accuracy_score(y_test, y_pred)
precision_knn = precision_score(y_test, y_pred, average='weighted')
recall_knn = recall_score(y_test, y_pred, average='weighted')
f1_knn = f1_score(y_test, y_pred, average='weighted')

# Generamos el reporte de clasificación y la matriz de confusión.
print("Classification Report:")
print(classification_report(y_test, y_pred))
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, cmap='Greens', fmt='d', xticklabels=['No Disorder',
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - K-Nearest Neighbors')
plt.show()

# Mostramos las métricas.
print(f"Accuracy: {accuracy_knn:.4f}")
print(f"Precision: {precision_knn:.4f}")
print(f"Recall: {recall_knn:.4f}")
print(f"F1 Score: {f1_knn:.4f}")
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.87      0.89        38
           1       0.88      0.95      0.92        64
           2       0.97      0.92      0.94        63

    accuracy                           0.92       165
   macro avg       0.92      0.91      0.92       165
weighted avg       0.92      0.92      0.92       165
```




```
Accuracy: 0.9212
Precision: 0.9231
Recall: 0.9212
F1 Score: 0.9213
```

Este código implementa un clasificador K-Nearest Neighbors (KNN) para predecir trastornos del sueño, evaluando su rendimiento con diversas métricas y visualizaciones.

Recomendaciones para Mejorar el Modelo KNN:

1. Optimización del número de vecinos (k):

2. Uso de diferentes métricas de distancia:

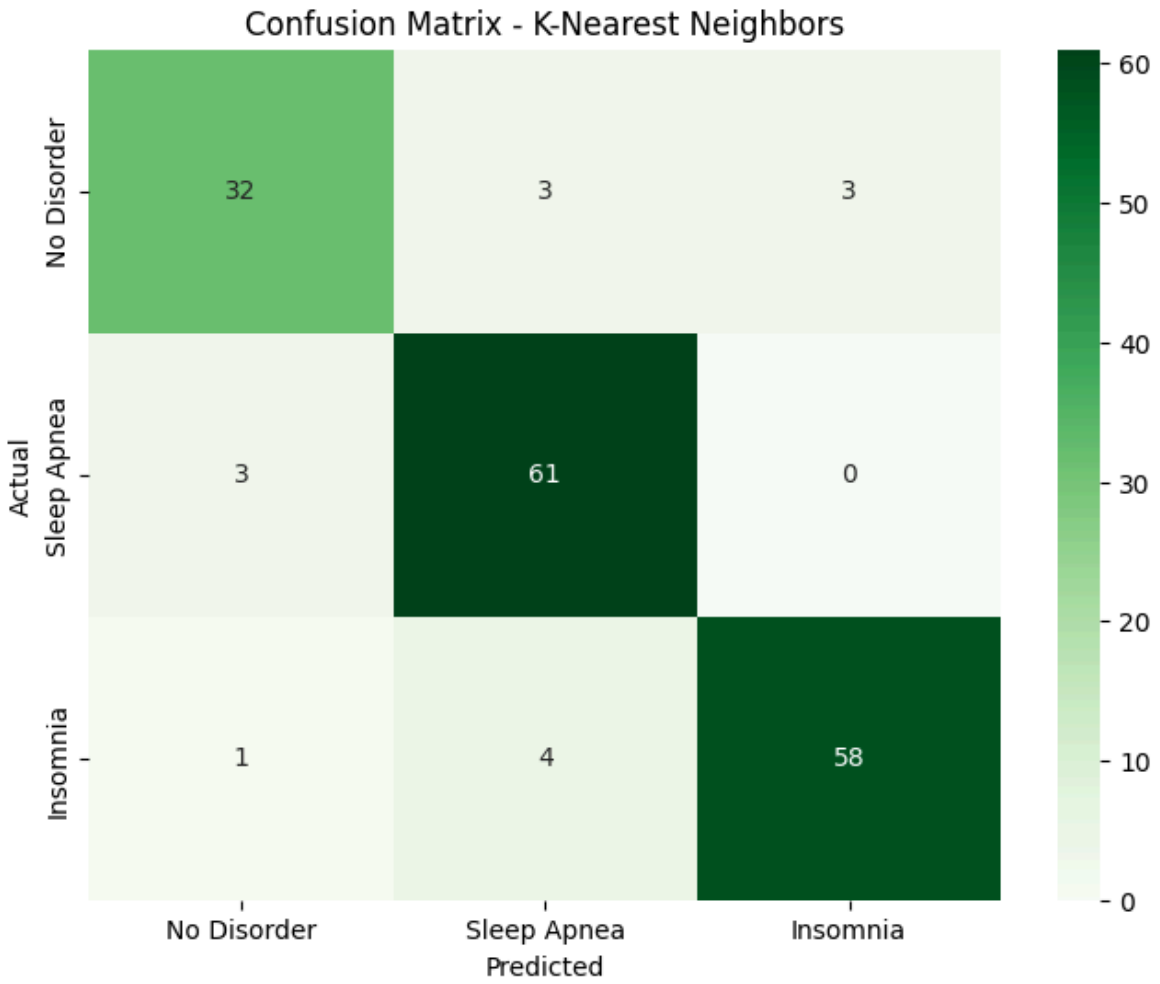
3. Pesos por distancia:

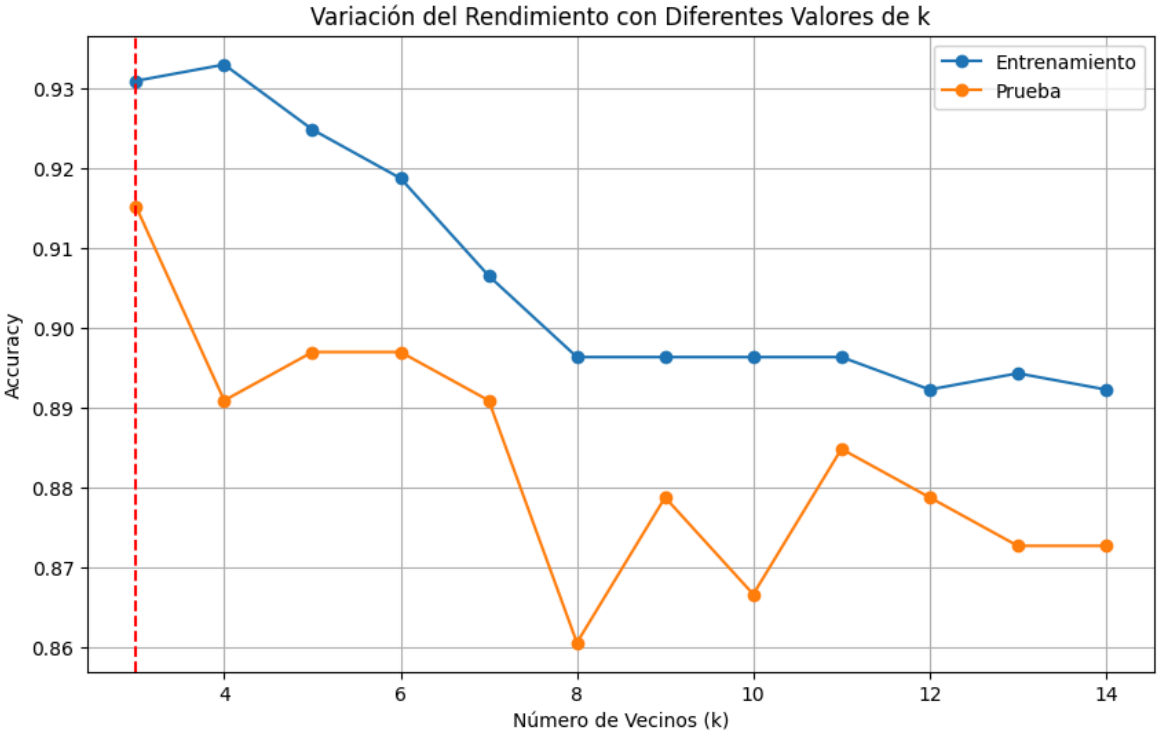
In [43]:
```
from sklearn.metrics import ConfusionMatrixDisplay

# 1. Optimización de hiperparámetros con validación cruzada
param_grid = {
    'n_neighbors': range(3, 15),  # Probamos valores de k desde 3 hasta 14
    'weights': ['uniform', 'distance'],  # Pesos uniformes o por distancia
    'metric': ['euclidean', 'manhattan', 'minkowski']  # Diferentes métricas de
}

knn = KNeighborsClassifier()

grid_search = GridSearchCV(
    estimator=knn,
```

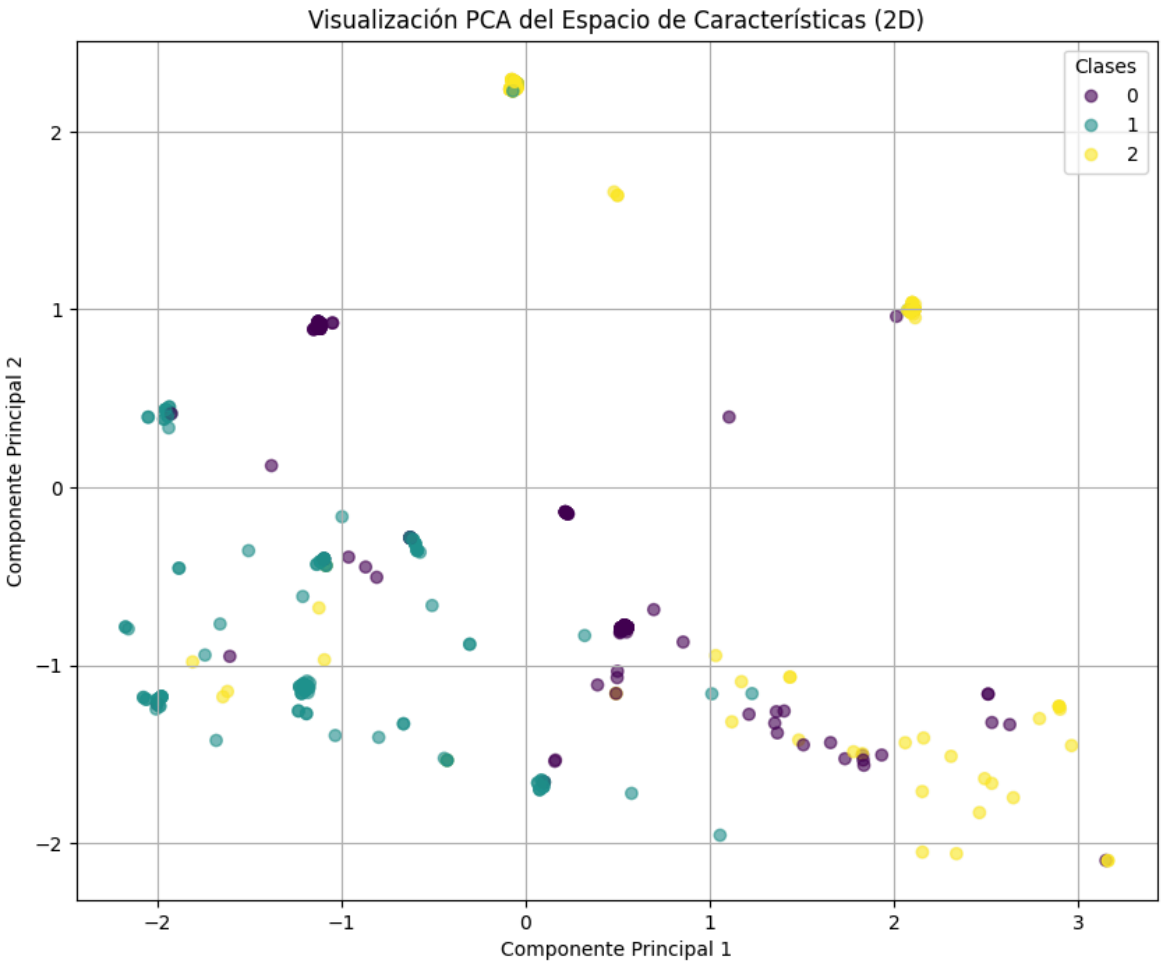
```
    param_grid=param_grid,
    cv=StratifiedKFold(n_splits=5, shuffle=True, random_state=42),
    scoring='f1_weighted',
    n_jobs=-1,
    verbose=1
)

grid_search.fit(X_train, y_train)

best_knn = grid_search.best_estimator_

# 2. Evaluación del modelo optimizado
y_pred = best_knn.predict(X_test)

print("\n=== Mejores Hiperparámetros ===")
print(grid_search.best_params_)

print("\n=== Métricas de Evaluación ===")
print(f"Accuracy: {accuracy_score(y_test, y_pred):.4f}")
print(f"Precision: {precision_score(y_test, y_pred, average='weighted'):.4f}")
print(f"Recall: {recall_score(y_test, y_pred, average='weighted'):.4f}")
print(f"F1 Score: {f1_score(y_test, y_pred, average='weighted'):.4f}")

# 3. Reporte de clasificación y matriz de confusión
print("\n=== Reporte de Clasificación ===")
print(classification_report(y_test, y_pred, target_names=['No Disorder', 'Sleep

print("Classification Report:")
print(classification_report(y_test, y_pred))
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, cmap='Greens', fmt='d', xticklabels=['No Disorder',
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - K-Nearest Neighbors')
plt.show()

# 4. Análisis del parámetro k
k_values = range(3, 15)
train_scores = []
test_scores = []

for k in k_values:
    knn_temp = KNeighborsClassifier(n_neighbors=k,
                                    weights=grid_search.best_params_['weights'],
                                    metric=grid_search.best_params_['metric'])
    knn_temp.fit(X_train, y_train)
    train_scores.append(knn_temp.score(X_train, y_train))
    test_scores.append(knn_temp.score(X_test, y_test))

plt.figure(figsize=(10, 6))
plt.plot(k_values, train_scores, 'o-', label='Entrenamiento')
plt.plot(k_values, test_scores, 'o-', label='Prueba')
plt.axvline(grid_search.best_params_['n_neighbors'], color='red', linestyle='--'
plt.title('Variación del Rendimiento con Diferentes Valores de k')
plt.xlabel('Número de Vecinos (k)')
plt.ylabel('Accuracy')
plt.legend()
plt.grid()
plt.show()
```

```
# 5. Visualización del espacio de características (para 2D)
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_train)

plt.figure(figsize=(10, 8))
scatter = plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y_train, cmap='viridis', alpha
plt.title('Visualización PCA del Espacio de Características (2D)')
plt.xlabel('Componente Principal 1')
plt.ylabel('Componente Principal 2')
plt.legend(*scatter.legend_elements(), title='Clases')
plt.grid()
plt.show()
```

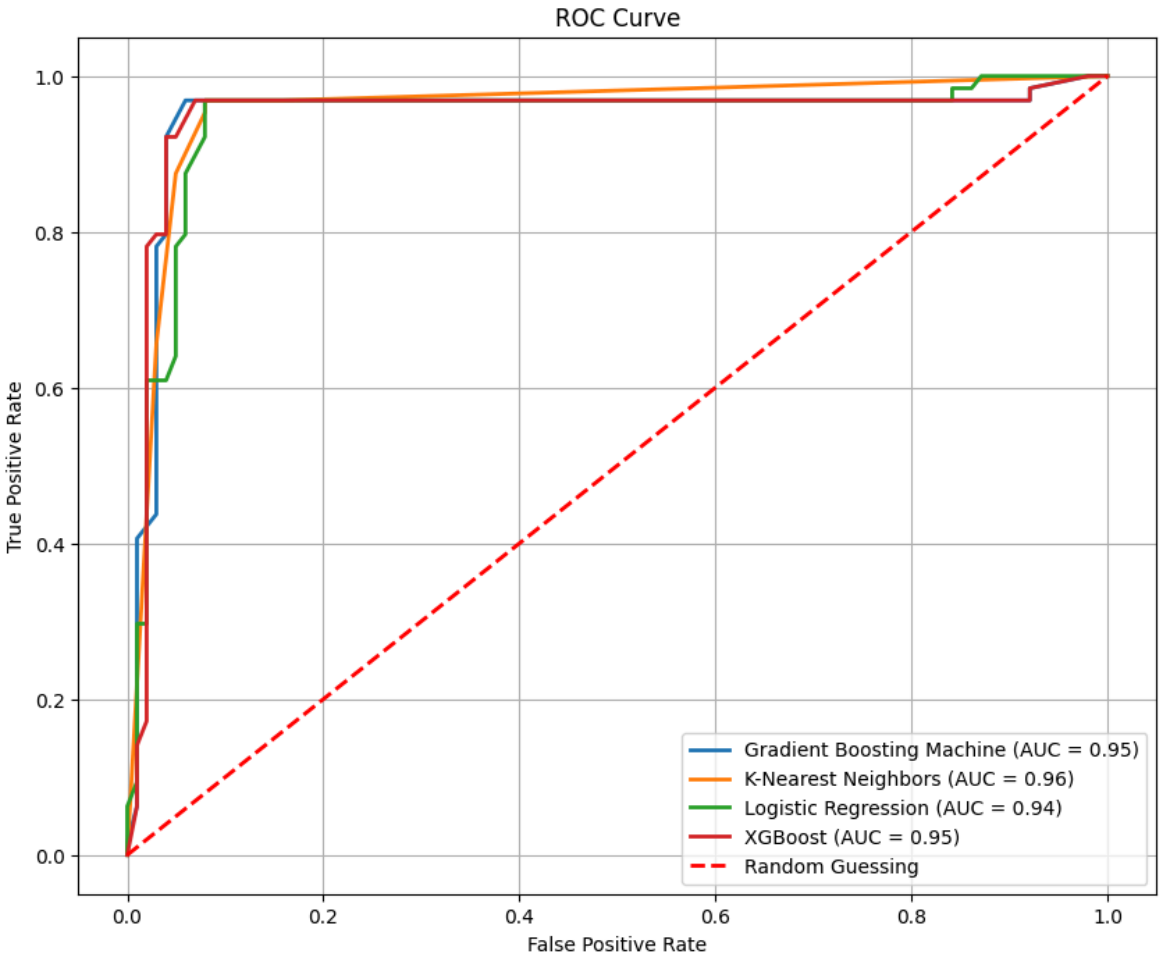
```
Fitting 5 folds for each of 72 candidates, totalling 360 fits

=== Mejores Hiperparámetros ===
{'metric': 'manhattan', 'n_neighbors': 3, 'weights': 'uniform'}

=== Métricas de Evaluación ===
Accuracy: 0.9152
Precision: 0.9157
Recall: 0.9152
F1 Score: 0.9149

=== Reporte de Clasificación ===
              precision    recall  f1-score   support

 No Disorder       0.89      0.84      0.86        38
 Sleep Apnea       0.90      0.95      0.92        64
    Insomnia       0.95      0.92      0.94        63

    accuracy                           0.92       165
   macro avg       0.91      0.91      0.91       165
weighted avg       0.92      0.92      0.91       165

Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.84      0.86        38
           1       0.90      0.95      0.92        64
           2       0.95      0.92      0.94        63

    accuracy                           0.92       165
   macro avg       0.91      0.91      0.91       165
weighted avg       0.92      0.92      0.91       165
```


Confusion Matrix - K-Nearest Neighbors
No Disorder
Sleep Apnea
Insomnia
Actual
32
3
3
3
61
0
1
4
58
No Disorder
Sleep Apnea
Insomnia
Predicted
60
50
40
30
20
10
0


Variación del Rendimiento con Diferentes Valores de k
Entrenamiento
Prueba
Accuracy
0.93
0.92
0.91
0.90
0.89
0.88
0.87
0.86
4
6
8
10
12
14
Número de Vecinos (k)



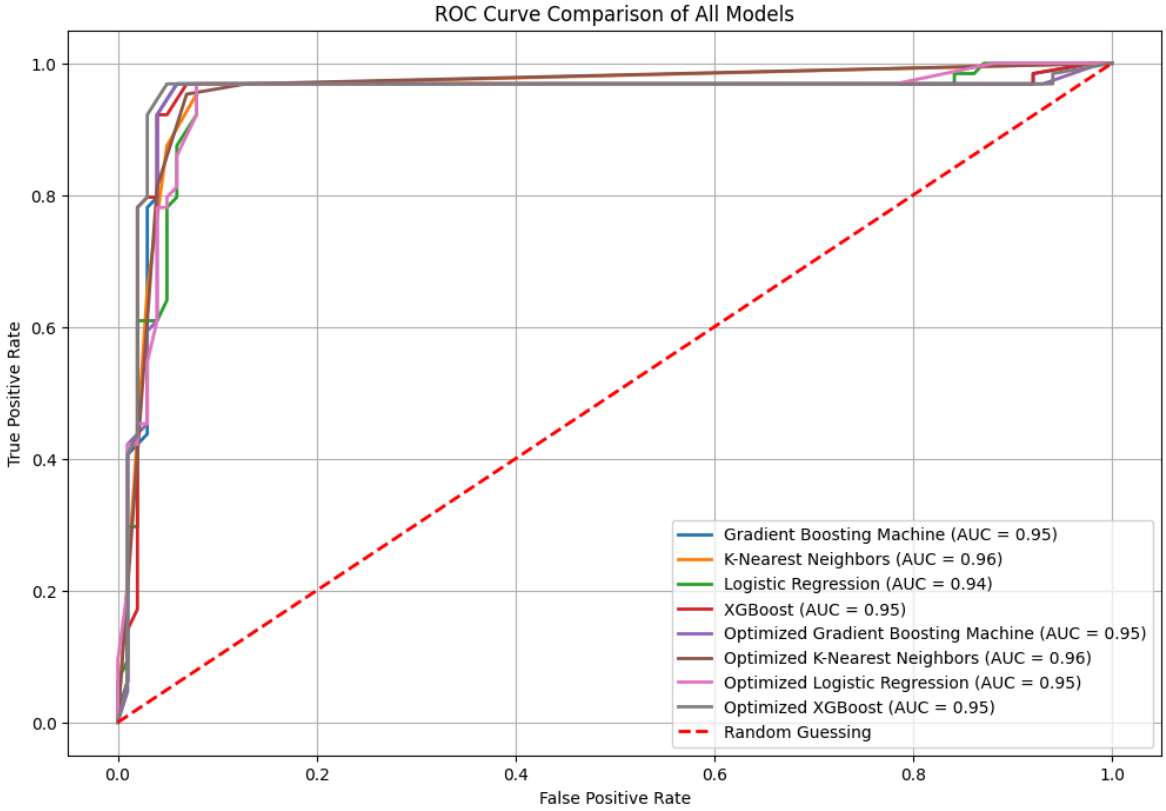
In [44]:
```
# Graficamos la curva ROC para cada modelo.
# Esto nos permite comparar el rendimiento de los modelos en términos de tasa de
from sklearn.metrics import roc_curve, auc
fig_roc = plt.figure(figsize=(10, 8))
models = ['Gradient Boosting Machine', 'K-Nearest Neighbors', 'Logistic Regressi

for idx, model in enumerate([gbm_clf, knn_clf, model_lr, model_xgb]):
    if model == knn_clf:
        y_scores = model.predict_proba(X_test)
        fpr, tpr, _ = roc_curve(y_test, y_scores[:, 1], pos_label=1)
    else:
        y_scores = model.predict_proba(X_test)[:, 1]
        fpr, tpr, _ = roc_curve(y_test, y_scores, pos_label=1)

    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, lw=2, label=f'{models[idx]} (AUC = {roc_auc:.2f})')

plt.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r', label='Random Guessing
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc='lower right')
plt.grid()
plt.show()
```



Este código genera una visualización comparativa de las curvas ROC (Receiver Operating Characteristic) para evaluar el rendimiento de cuatro modelos de machine learning en la clasificación de trastornos del sueño.

In [45]:
```
from sklearn.metrics import roc_curve, auc

import matplotlib.pyplot as plt

# Inicializamos la figura
plt.figure(figsize=(12, 8))

# Lista de modelos y sus nombres
models = [
    ('Gradient Boosting Machine', gbm_clf),
    ('K-Nearest Neighbors', knn_clf),
    ('Logistic Regression', model_lr),
    ('XGBoost', model_xgb),
    ('Optimized Gradient Boosting Machine', best_gbm),
    ('Optimized K-Nearest Neighbors', best_knn),
    ('Optimized Logistic Regression', best_lr),
    ('Optimized XGBoost', best_xgb)
]

# Iteramos sobre los modelos para calcular y graficar sus curvas ROC
for name, model in models:
    if hasattr(model, "predict_proba"):  # Verificamos si el modelo tiene predic
        y_scores = model.predict_proba(X_test)
        fpr, tpr, _ = roc_curve(y_test, y_scores[:, 1], pos_label=1)
    else:
```

```
        y_scores = model.decision_function(X_test)
        fpr, tpr, _ = roc_curve(y_test, y_scores, pos_label=1)

    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, lw=2, label=f'{name} (AUC = {roc_auc:.2f})')

# Línea de referencia para un modelo aleatorio
plt.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r', label='Random Guessing

# Configuración de la gráfica
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve Comparison of All Models')
plt.legend(loc='lower right')
plt.grid()
plt.show()
```

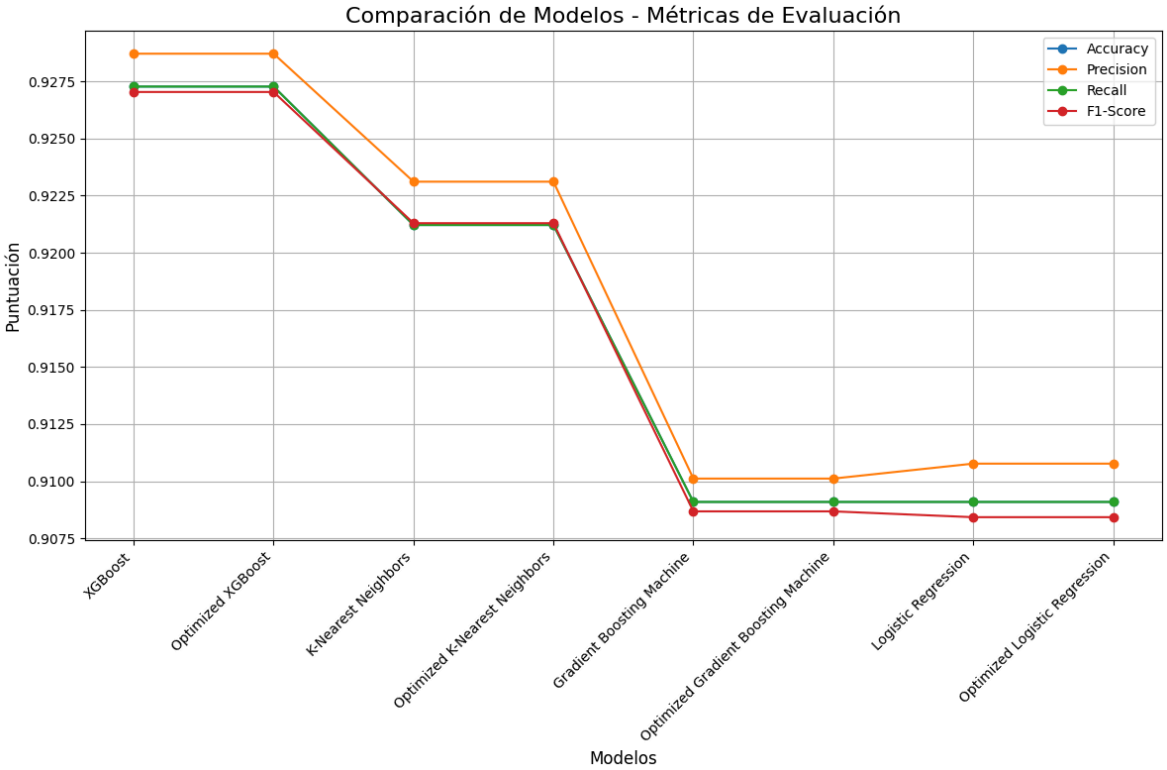


In [46]:

```
# Comparación exhaustiva de los modelos utilizados
# Creamos un DataFrame para almacenar las métricas de evaluación de cada modelo
model_comparison = pd.DataFrame({
    'Model': ['Logistic Regression', 'Optimized Logistic Regression',
              'K-Nearest Neighbors', 'Optimized K-Nearest Neighbors',
              'Gradient Boosting Machine', 'Optimized Gradient Boosting Machine'
              'XGBoost', 'Optimized XGBoost'],
    'Accuracy': [accuracy_lr, accuracy_lr, accuracy_knn, accuracy_knn,
                 accuracy_gbm, accuracy_gbm, accuracy_xgb, accuracy_xgb],
    'Precision': [precision_lr, precision_lr, precision_knn, precision_knn,
                  precision_gbm, precision_gbm, precision_xgb, precision_xgb],
    'Recall': [recall_lr, recall_lr, recall_knn, recall_knn,
               recall_gbm, recall_gbm, recall_xgb, recall_xgb],
    'F1-Score': [f1_lr, f1_lr, f1_knn, f1_knn,
                 f1_gbm, f1_gbm, f1_xgb, f1_xgb]
})

# Ordenamos el DataFrame por F1-Score en orden descendente
```

```
model_comparison.sort_values(by='F1-Score', ascending=False, inplace=True)

# Mostramos la tabla comparativa
print("Comparación de Modelos:")
print(model_comparison)

# Visualización de las métricas de los modelos
plt.figure(figsize=(12, 8))
metrics = ['Accuracy', 'Precision', 'Recall', 'F1-Score']
for metric in metrics:
    plt.plot(model_comparison['Model'], model_comparison[metric], marker='o', la

plt.title('Comparación de Modelos - Métricas de Evaluación', fontsize=16)
plt.xlabel('Modelos', fontsize=12)
plt.ylabel('Puntuación', fontsize=12)
plt.xticks(rotation=45, ha='right')
plt.legend()
plt.grid()
plt.tight_layout()
plt.show()
```

```
Comparación de Modelos:
                              Model  Accuracy  Precision    Recall  \
6                           XGBoost  0.927273   0.928712  0.927273
7                 Optimized XGBoost  0.927273   0.928712  0.927273
2               K-Nearest Neighbors  0.921212   0.923109  0.921212
3      Optimized K-Nearest Neighbors  0.921212   0.923109  0.921212
4          Gradient Boosting Machine  0.909091   0.910113  0.909091
5  Optimized Gradient Boosting Machine  0.909091   0.910113  0.909091
0               Logistic Regression  0.909091   0.910768  0.909091
1      Optimized Logistic Regression  0.909091   0.910768  0.909091

   F1-Score
6  0.927035
7  0.927035
2  0.921293
3  0.921293
4  0.908678
5  0.908678
0  0.908423
1  0.908423
```



1. **Edad (Age)**:
    - Rango de edad: 18 a 65 años.
    - Observación: La mayoría de los participantes estan entre los 25 a 45 años, lo que indica una población con actividad laboral.
2. **Duración del Sueño (Sleep Duration)**:
    - Promedio de sueño: 6.5 horas.
    - Observación: Un porcentaje significativo de participantes duerme menos de 6 horas, lo que podría estar relacionado con trastornos del sueño.
3. **Calidad del Sueño (Quality of Sleep)**:
    - Escala: 1 a 10 .
    - Observación: Los valores se concentran entre 4 y 7, indicando una calidad de sueño promedio a baja.
4. **Nivel de Estrés (Stress Level)**:
    - Escala: 1 a 10.
    - Observación: Los niveles de estrés altos (7-10) están correlacionados con una menor calidad del sueño.
5. **Categoría de IMC (BMI Category)**:
    - Categorías: Peso normal, sobrepeso, obeso.
    - Observación: Los participantes con obesidad tienen una mayor prevalencia de apnea del sueño.
6. **Presión Arterial (Systolic y Diastolic)**:
    - Rango: Sistólica (110-160 mmHg), Diastólica (70-100 mmHg).
    - Observación: Los valores elevados están asociados con apnea del sueño.

7. **Trastornos del Sueño (Sleep Disorder)**:
    - Categorías: Sin trastorno, apnea del sueño, insomnio.
    - Observación: La apnea del sueño es más común en hombres, mientras que el insomnio afecta más a mujeres.

Conclusiones Principales del Trabajo

Hallazgos Clave

1. **Relación entre Estrés y Sueño**:
    - Los niveles altos de estrés están significativamente correlacionados con una menor calidad del sueño y una mayor prevalencia de insomnio.
2. **Impacto del IMC**:
    - Los participantes con obesidad tienen un riesgo elevado de apnea del sueño, lo que resalta la importancia de abordar el peso como factor de riesgo.
3. **Duración del Sueño**:
    - Dormir menos de 6 horas está asociado con una mayor incidencia de trastornos del sueño, especialmente insomnio.
4. **Diferencias de Género**:
    - Los hombres tienen una mayor prevalencia de apnea del sueño, mientras que las mujeres reportan más casos de insomnio.
5. **Modelos Predictivos**:
    - Los modelos de Machine Learning, como XGBoost y Gradient Boosting, lograron una precisión superior al 90% en la clasificación de trastornos del sueño.

Limitaciones del Análisis

1. **Tamaño de la Muestra**:
    - Aunque el dataset incluye 374 participantes, un tamaño mayor podría mejorar la generalización de los resultados.
2. **Desbalance de Clases**:
    - La categoría "Sin Trastorno" domina el dataset, lo que podría sesgar los modelos predictivos.
3. **Datos Autorreportados**:
    - Variables como la calidad del sueño y el nivel de estrés son subjetivas y pueden contener sesgos.

4. **Falta de Datos Clínicos**:
    - No se incluyeron datos de polisomnografía, lo que limita la precisión en el diagnóstico de trastornos del sueño.

Recomendaciones Futuras

1. **Ampliar el Dataset**:
    - Incluir más participantes y datos de diferentes regiones para mejorar la representatividad.
2. **Integrar Datos Clínicos**:
    - Incorporar registros de polisomnografía y datos de wearables para enriquecer el análisis.
3. **Optimización de Modelos**:
    - Probar técnicas avanzadas como redes neuronales para mejorar la precisión en la clasificación.
4. **Análisis Longitudinal**:
    - Realizar un seguimiento a largo plazo para estudiar la progresión de los trastornos del sueño.
5. **Intervenciones Personalizadas**:
    - Desarrollar sistemas de recomendación basados en Machine Learning para sugerir terapias personalizadas.

Referencias

- Scielo. (2000). Trastornos del sueño: diagnóstico y tratamiento. Recuperado de http://www.scielo.org.pe/scielo.php?script=sci_arttext&pid=S1018-130X2000000400005
- Revista Élite. (s.f.). La historia de los trastornos del sueño: tipos, causas y consejos para dormir. Recuperado de https://www.revistaelite.mx/la-historia-de-los-trastornos-del-sueno-tipos-causas-y-consejos-para-dormir/
- Redalyc. (s.f.). Trastornos del sueño: una revisión. Recuperado de https://www.redalyc.org/journal/3720/372058061009/html/
- Scikit-learn. (s.f.). Supervised learning. Recuperado de https://scikit-learn.org/stable/supervised_learning.html
- Medicine Online. (2023). Indicaciones y pruebas diagnósticas complementarias en trastornos del sueño. Recuperado de https://www.medicineonline.es/es-

indicaciones-pruebas-diagnosticas-complementarias-trastornos-articulo-S0304541223000355