Proyecto

May 26, 2025

[90]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
from scipy.stats import chi2_contingency
import prince
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV, StratifiedKFold
from sklearn.metrics import classification_report
from sklearn.preprocessing import StandardScaler, OneHotEncoder, LabelEncoder
from sklearn.compose import ColumnTransformer
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
```

1 Presentación

Integrantes:
- Antonio Anselmi, Miguel Maximiliano (20200118).

[71]:
```
datos = pd.read_csv("ObesityDataSet_raw_and_data_sinthetic.csv",sep=",")
```

[72]:
```
datos.rename({"family_history_with_overweight":"fwo"},inplace=True,axis=1)
```

Resultado: No hay datos faltantes en ningun campo, y hay variables cualitativas y cuantitativas

¿Mientras mayor es la cantidad de comidas principales, menor es el control de consumo calórico?

2 Análisis Exploratorio de Datos

2.1 Descripción general del dataset

Instancias: 2111 personas
Características : 17 atributos + 1 variable objetivo
Objetivo : Clasificar el nivel de obesidad de una persona

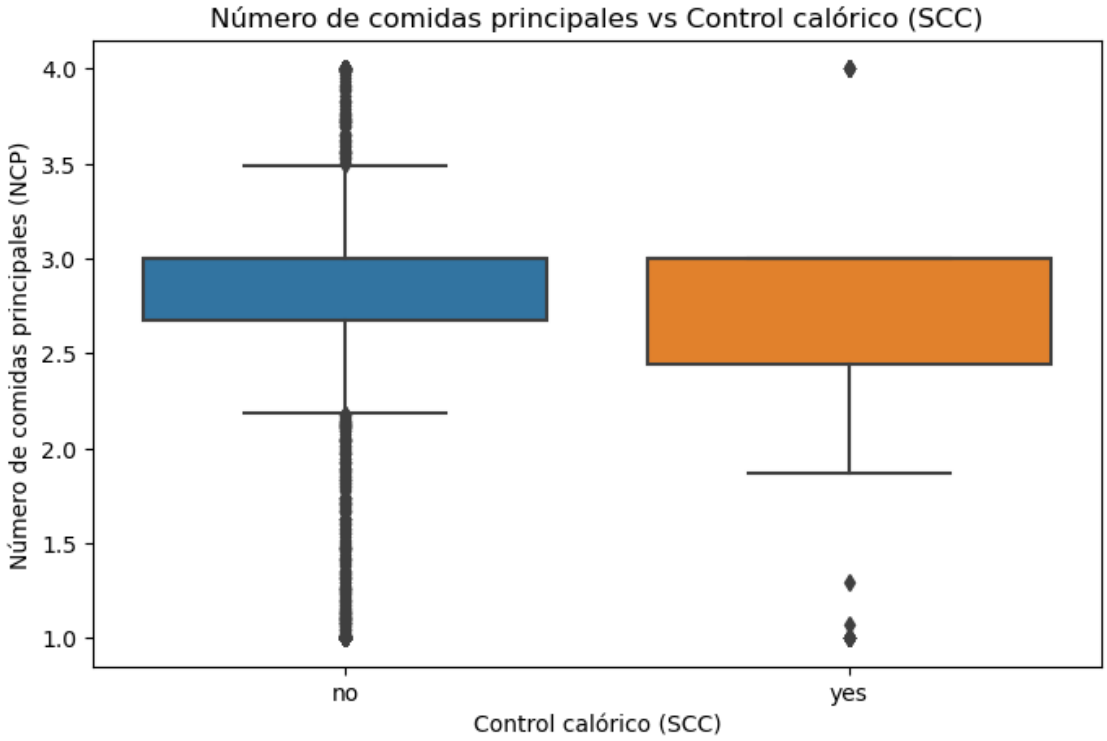
[73]:
```
pd.DataFrame(datos.groupby('SCC')['NCP'].describe()["count"])
```

[73]:

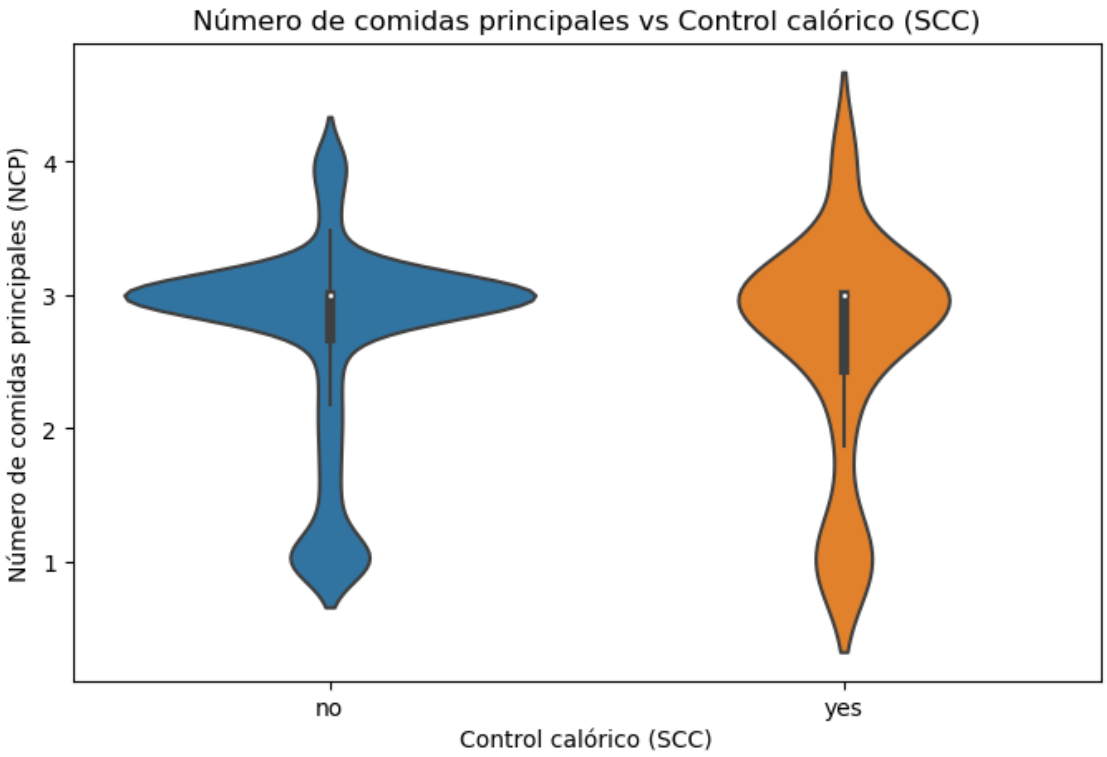
SCC	count
no	2015.0
yes	96.0

La mayoría de los entrevistados no controlan las calorías que consumen

[74]:
```
plt.figure(figsize=(8, 5))
sns.boxplot(data=datos, x='SCC', y='NCP')
plt.title('Número de comidas principales vs Control calórico (SCC)')
plt.xlabel('Control calórico (SCC)')
plt.ylabel('Número de comidas principales (NCP)')
plt.show()
```

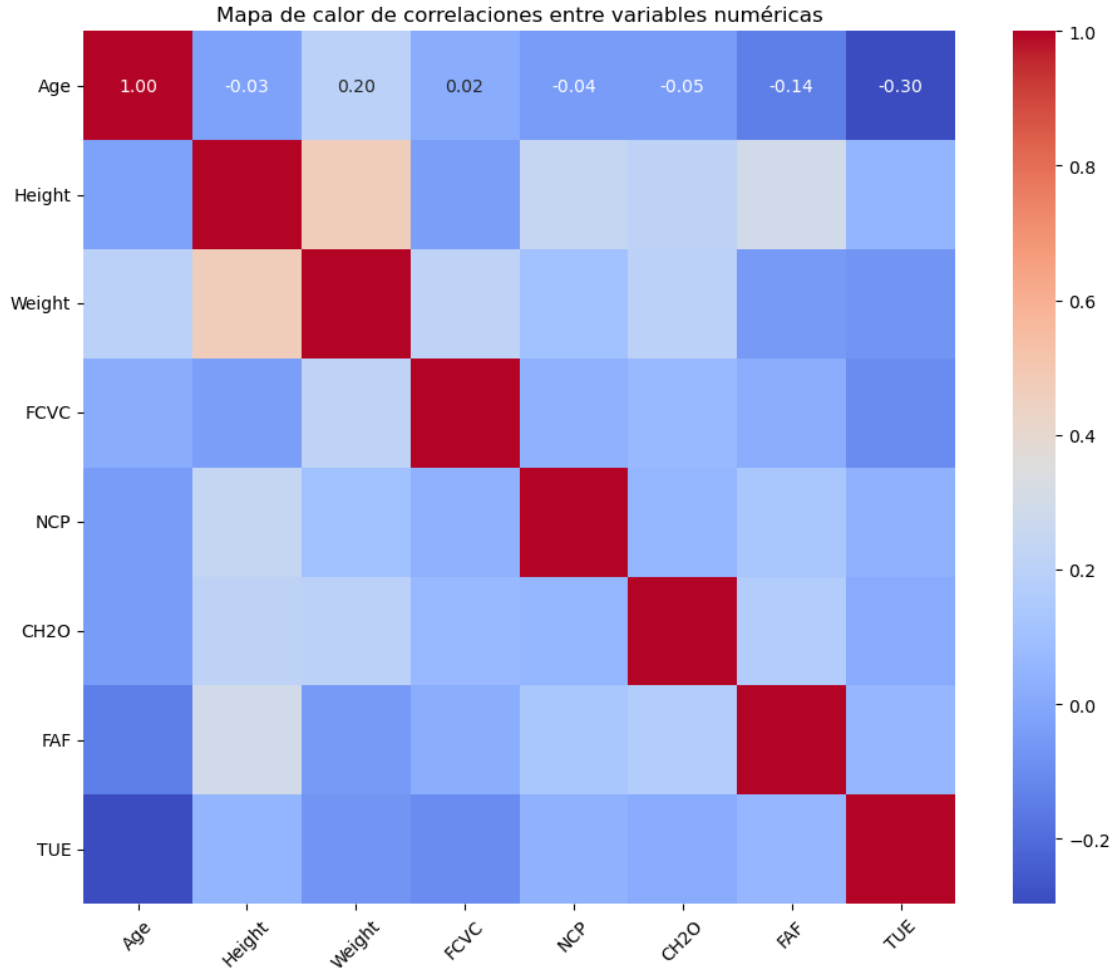


[75]:
```
plt.figure(figsize=(8, 5))
sns.violinplot(data=datos, x='SCC', y='NCP')
plt.title('Número de comidas principales vs Control calórico (SCC)')
plt.xlabel('Control calórico (SCC)')
plt.ylabel('Número de comidas principales (NCP)')
plt.show()
```



Podemos apreciar que las personas que **si controlan** su consumo de calorías, tienen un mayor rango en la decisión del **Número de comidas principales**. En cambio, los que **no controlan** su consumo de calorías, tienden **en mayoría** , comer **solamente** las 3 comidas principales del día (valor impuesto por la sociedad).

[76]:
```
# Filtramos solo las variables numéricas
datos_numericos = datos.select_dtypes(include=['float64', 'int64'])

# Calculamos la matriz de correlación
matriz_corr = datos_numericos.corr()

# Dibujamos el mapa de calor
plt.figure(figsize=(10, 8))
sns.heatmap(matriz_corr, annot=True, cmap='coolwarm', fmt=".2f", square=True)
plt.title('Mapa de calor de correlaciones entre variables numéricas')
plt.xticks(rotation=45)
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()
```



[77]:
```
# Matriz de correlación
corr_matrix = datos_numericos.corr().abs()

# Eliminamos la diagonal (autocorrelaciones)
mask = np.triu(corr_matrix, k=1)

# Convertimos en tabla y filtramos correlaciones altas (> 0.7)
high_corr_pairs = (
    corr_matrix.where(mask > 0)
    .stack()
    .reset_index()
    .rename(columns={'level_0': 'Variable 1', 'level_1': 'Variable 2', 0:␣
 ↪'Correlación'})
    .sort_values(by='Correlación', ascending=False)
)
```

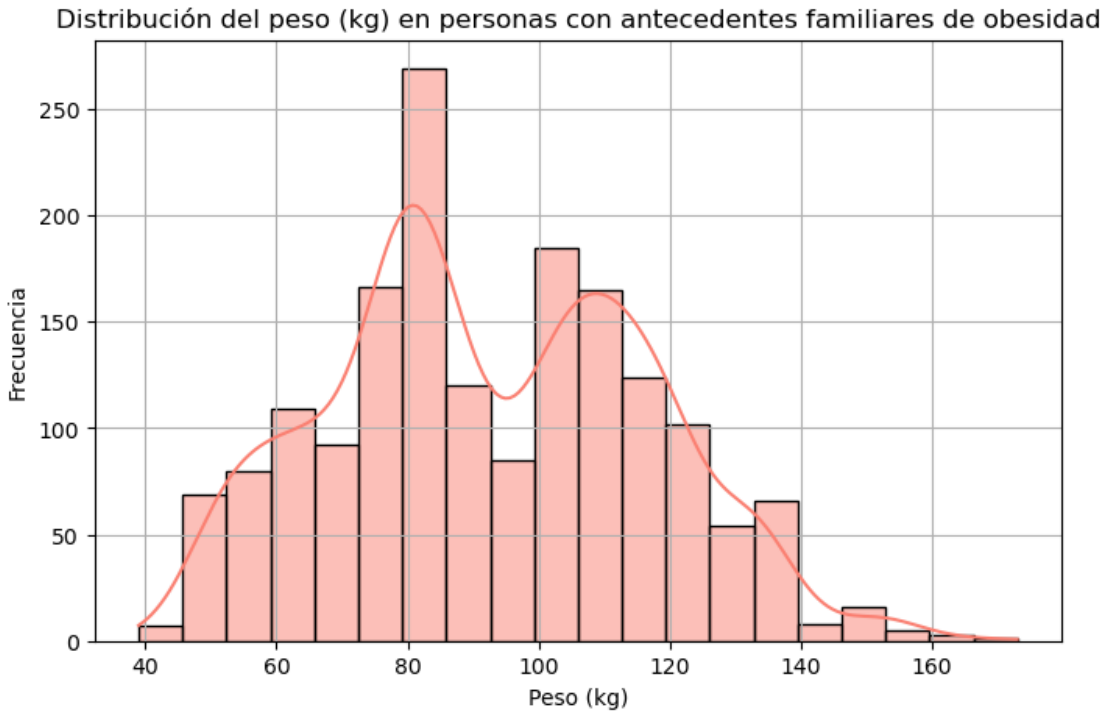
```
# Filtrar pares con correlación alta
umbral = 0.4
high_corr_pairs = high_corr_pairs[high_corr_pairs['Correlación'] > umbral]
high_corr_pairs
```

[77]:

	Variable 1	Variable 2	Correlación
7	Height	Weight	0.463136

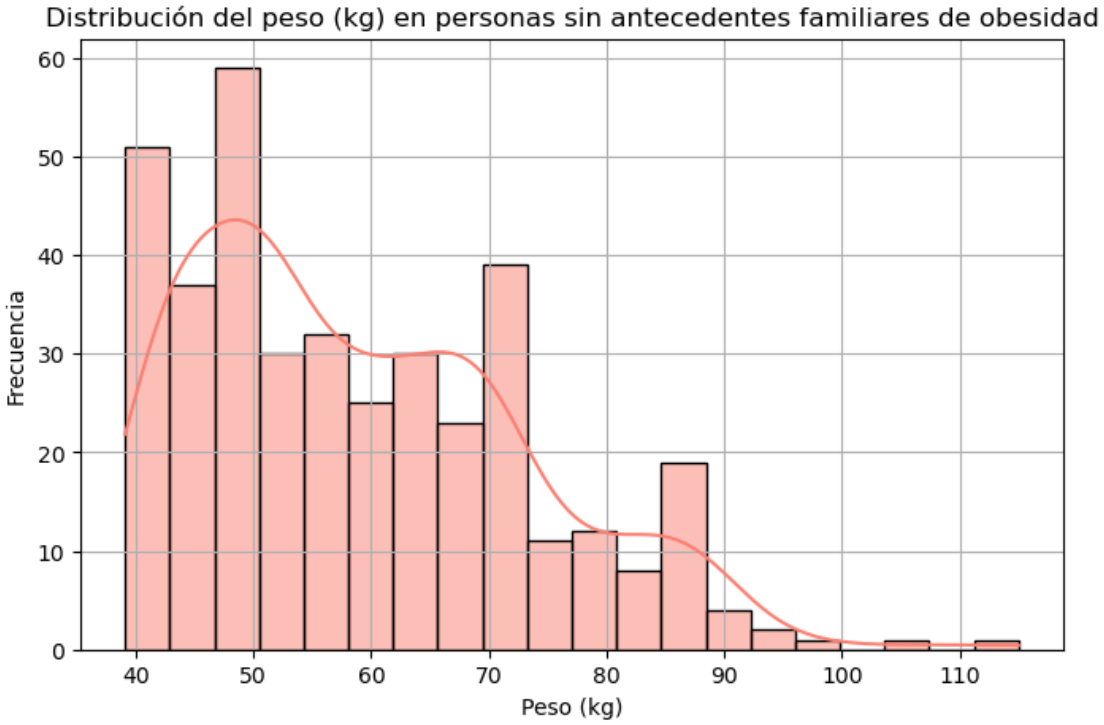
Hay **correlación alta** entre el Peso y la talla de la persona entrevistada.

[78]:
```
# Filtrar solo los que tienen antecedentes familiares de obesidad
peso_con_antecedentes = datos[datos['fwo'] == 'yes']['Weight']
plt.figure(figsize=(8, 5))
sns.histplot(peso_con_antecedentes, bins=20, kde=True, color='salmon')
plt.title('Distribución del peso (kg) en personas con antecedentes familiares␣
↪de obesidad')
plt.xlabel('Peso (kg)')
plt.ylabel('Frecuencia')
plt.grid(True)
plt.show()
```



[79]:
```
# Filtrar solo los que tienen antecedentes familiares de obesidad
peso_con_antecedentes = datos[datos['fwo'] == 'no']['Weight']
plt.figure(figsize=(8, 5))
```

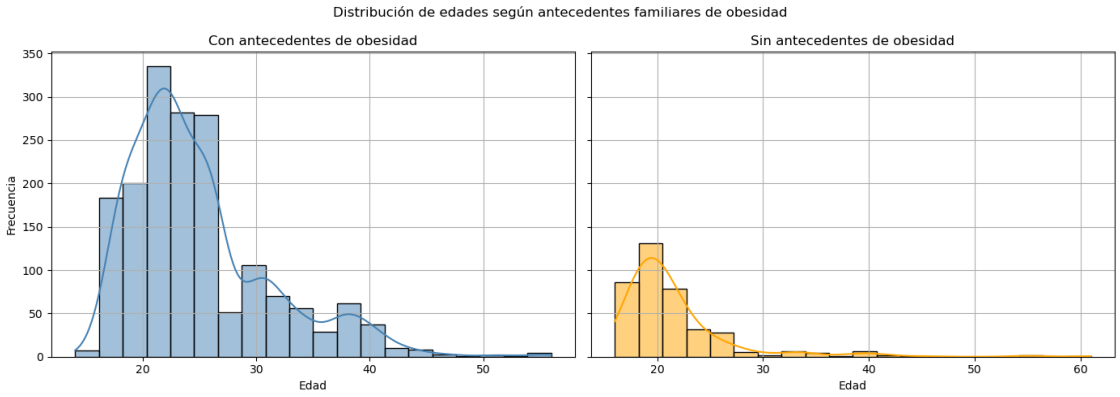
```
sns.histplot(peso_con_antecedentes, bins=20, kde=True, color='salmon')
plt.title('Distribución del peso (kg) en personas sin antecedentes familiares␣
 ↪de obesidad')
plt.xlabel('Peso (kg)')
plt.ylabel('Frecuencia')
plt.grid(True)
plt.show()
```



La distribución del peso de las personas que **NO tienen antecedentes familiares con obesidad** , tienen un peso con **asimetría positiva**, es decir que tienden a tener menor peso a comparación de los que **SI tienen antecedentes familiares con obesidad**. Sin embargo es raro que el peso de 40kg sea muy frecuente en el grupo sin antecedentes. Veamos:

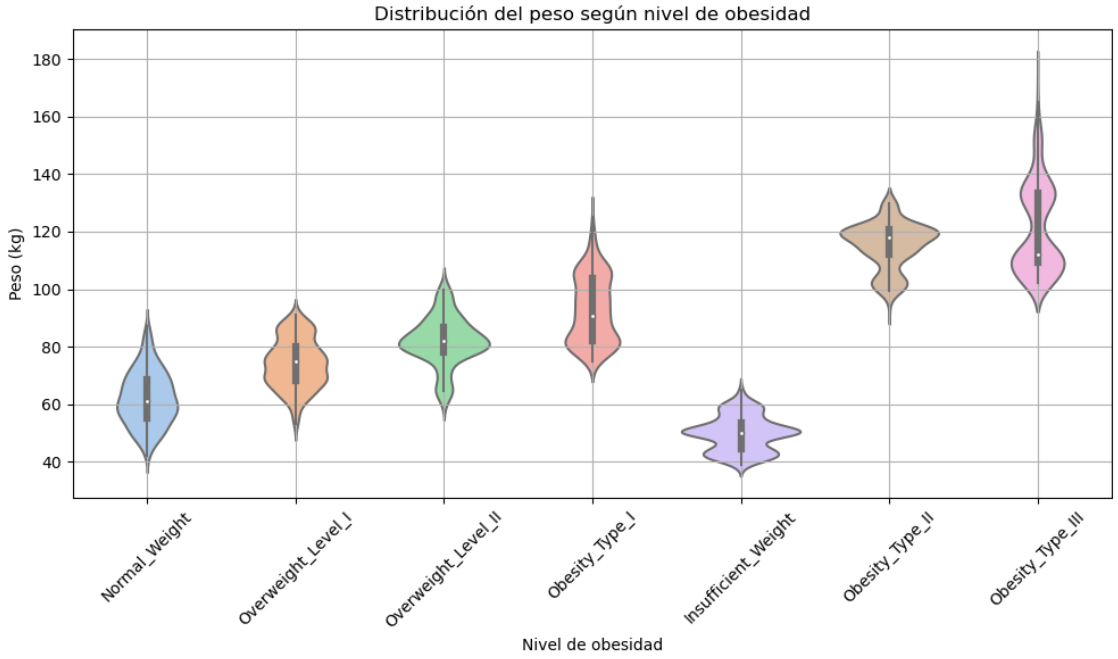
[80]:
```
# Filtrar datos
edad_con_antecedentes = datos[datos['fwo'] == 'yes']['Age']
edad_sin_antecedentes = datos[datos['fwo'] == 'no']['Age']

# Crear subplots: 1 fila, 2 columnas
fig, axes = plt.subplots(1, 2, figsize=(14, 5), sharey=True)

# Histograma 1: con antecedentes
sns.histplot(edad_con_antecedentes, bins=20, kde=True, color='steelblue',␣
 ↪ax=axes[0])
```

```
axes[0].set_title('Con antecedentes de obesidad')
axes[0].set_xlabel('Edad')
axes[0].set_ylabel('Frecuencia')
axes[0].grid(True)

# Histograma 2: sin antecedentes
sns.histplot(edad_sin_antecedentes, bins=20, kde=True, color='orange',␣
 ↪ax=axes[1])
axes[1].set_title('Sin antecedentes de obesidad')
axes[1].set_xlabel('Edad')
axes[1].set_ylabel('')
axes[1].grid(True)

# Título general
plt.suptitle('Distribución de edades según antecedentes familiares de obesidad')
plt.tight_layout()
plt.show()
```



Podemos observar que hay más caso de personas **con antecedentes familiares de obesidad** en este dataset. Tambien hay mayor variación de edad en este grupo respecto al otro.

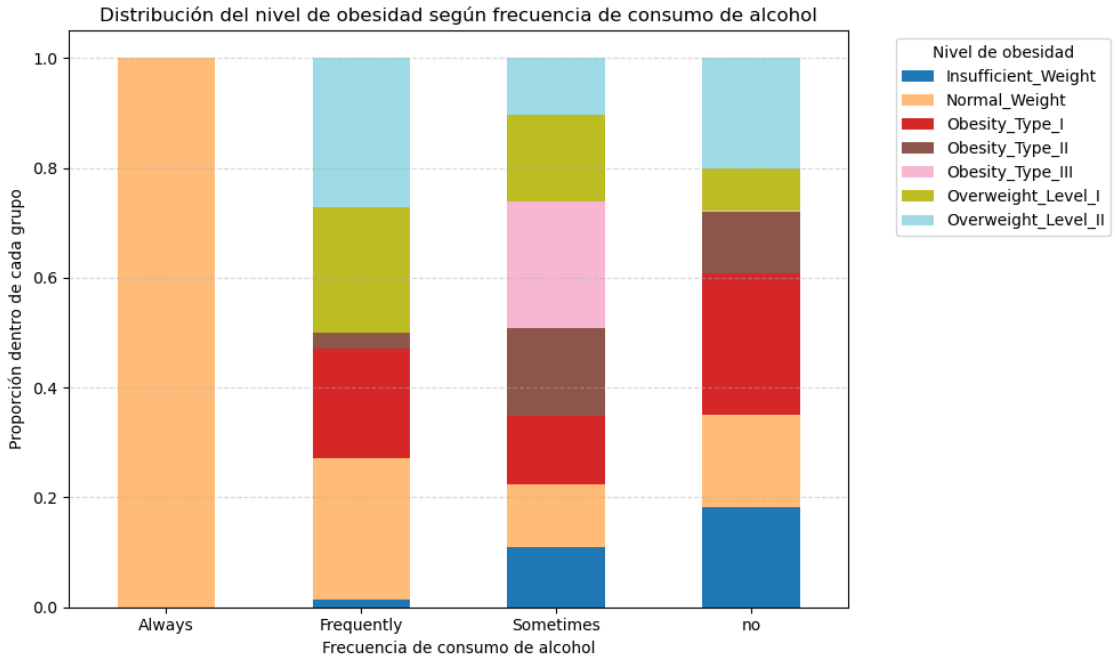
[81]:
```
plt.figure(figsize=(10, 6))
sns.violinplot(data=datos, x='NObeyesdad', y='Weight', palette='pastel')
plt.title('Distribución del peso según nivel de obesidad')
plt.xlabel('Nivel de obesidad')
plt.ylabel('Peso (kg)')
plt.xticks(rotation=45)
plt.grid(True)
plt.tight_layout()
plt.show()
```



El **peso** es una **variable importante** para entrenar nuestro modelo de clasificación. Podemos observar que hay diferencias entre los pesos de los niveles

[82]:
```
datos_renombrado = datos.rename(columns={
    'CALC': 'Frecuencia de consumo de alcohol',
    'NObeyesdad': 'Nivel de obesidad'
})

# Crosstab normalizado por fila (frecuencia relativa por grupo)
tabla = pd.crosstab(datos_renombrado['Frecuencia de consumo de alcohol'],
                    datos_renombrado['Nivel de obesidad'],
                    normalize='index')

# Graficar
ax = tabla.plot(kind='bar', stacked=True, figsize=(10, 6), colormap='tab20')

# Mejoras visuales
plt.title('Distribución del nivel de obesidad según frecuencia de consumo de␣
 ↪alcohol')
plt.xlabel('Frecuencia de consumo de alcohol')
plt.ylabel('Proporción dentro de cada grupo')
plt.xticks(rotation=0)
plt.legend(title='Nivel de obesidad', bbox_to_anchor=(1.05, 1), loc='upper␣
 ↪left')
plt.tight_layout()
```

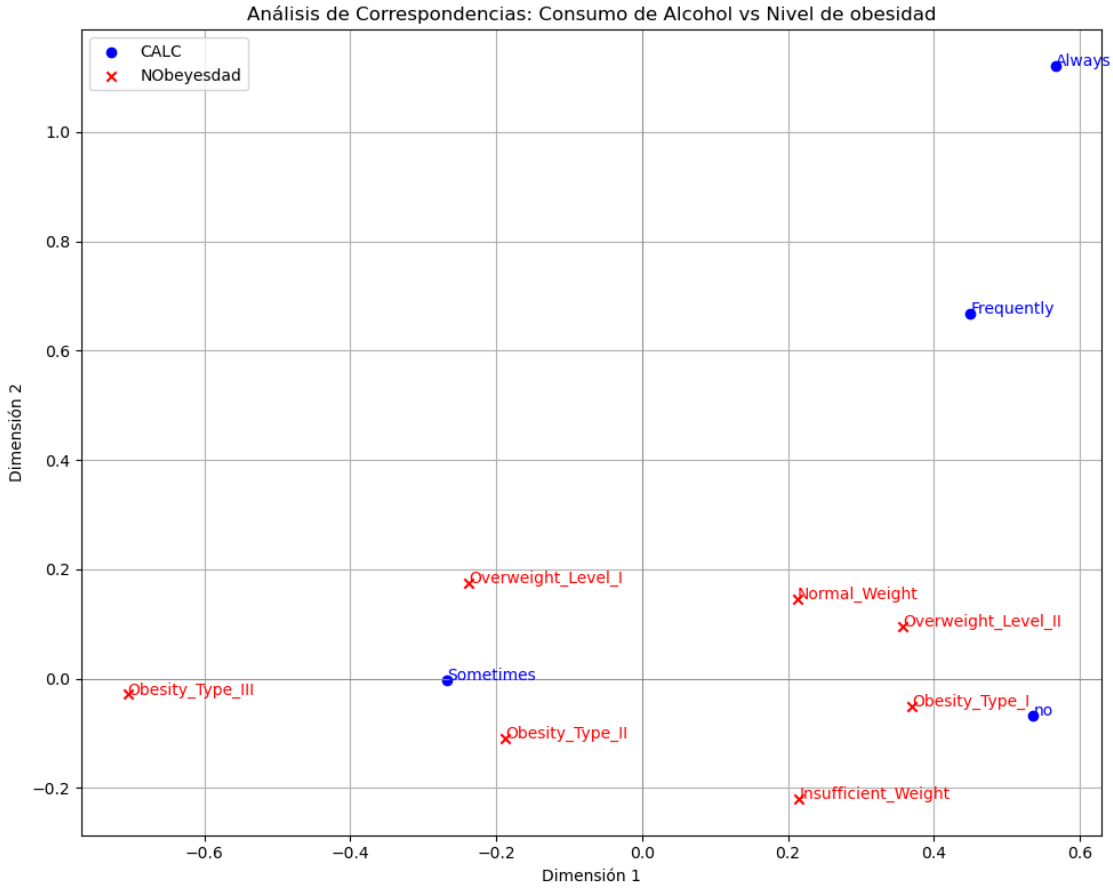
```
plt.grid(axis='y', linestyle='--', alpha=0.5)
plt.show()
```



[83]:
```
# Tabla de contingencia entre CALC y NObeyesdad
tabla = pd.crosstab(datos['CALC'], datos['NObeyesdad'])

# Test de Chi-cuadrado
chi2, p, dof, expected = chi2_contingency(tabla)

# Resultados
print("Estadístico Chi²:", round(chi2, 3))
print("Grados de libertad:", dof)
print("Valor p:", round(p, 4))

if p < 0.05:
    print("  Hay una relación significativa entre consumo de alcohol y nivel de␣
 ↪obesidad (se rechaza H0)")
else:
    print("  No hay evidencia suficiente para afirmar que están asociadas (no␣
 ↪se rechaza H0)")
```

```
Estadístico Chi²: 338.578
Grados de libertad: 18
Valor p: 0.0
  Hay una relación significativa entre consumo de alcohol y nivel de obesidad
(se rechaza H0)
```

[84]:
```
# Crear la tabla de contingencia
tabla = pd.crosstab(datos['CALC'], datos['NObeyesdad'])

# Realizar el Análisis de Correspondencias (CA)
ca = prince.CA(n_components=2, random_state=42)
ca = ca.fit(tabla)

# Obtener coordenadas de filas y columnas
coord_filas = ca.row_coordinates(tabla)
coord_columnas = ca.column_coordinates(tabla)

# Graficar
plt.figure(figsize=(10, 8))

# Filas (categorías de CALC)
plt.scatter(coord_filas[0], coord_filas[1], c='blue', label='CALC')
for i, label in enumerate(coord_filas.index):
    plt.text(coord_filas.iloc[i, 0], coord_filas.iloc[i, 1], label,␣
 ↪color='blue')

# Columnas (categorías de NObeyesdad)
plt.scatter(coord_columnas[0], coord_columnas[1], c='red', label='NObeyesdad',␣
 ↪marker='x')
for i, label in enumerate(coord_columnas.index):
    plt.text(coord_columnas.iloc[i, 0], coord_columnas.iloc[i, 1], label,␣
 ↪color='red')

plt.axhline(0, color='gray', lw=0.5)
plt.axvline(0, color='gray', lw=0.5)
plt.title('Análisis de Correspondencias: Consumo de Alcohol vs Nivel de␣
 ↪obesidad')
plt.xlabel('Dimensión 1')
plt.ylabel('Dimensión 2')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



Interesante que los niveles de obesidad más altos no esten necesariamente asociados con el consumo de alcohol en alta frecuencia.

[85]:
```
datos.groupby(["NObeyesdad"],as_index=True)["NObeyesdad"].count()/2111
```

[85]:
```
NObeyesdad
Insufficient_Weight    0.128849
Normal_Weight          0.135955
Obesity_Type_I         0.166272
Obesity_Type_II        0.140692
Obesity_Type_III       0.153482
Overweight_Level_I     0.137376
Overweight_Level_II    0.137376
Name: NObeyesdad, dtype: float64
```

El target esta completamente **balanceado**.

3 Modelamiento

Probaremos los Algoritmos **KNN** , **Random Forest**, **Regresión Logística** y lo tunearemos cada modelo mediante **validación cruzada** de 10 capas, usaremos **F1-Score** para elegir el mejor modelo ya que pondera **Precisión** y **Recall** para no subestimar categorías.
Tambien repartiremos los datos en proporción de **70:30** para **Entrenamiento** y **Validación** respectivamente.
KNN:
- Se comparará el entrenamiento con **3,5,7** vecinos.
- Se comparará el entrenamiento con pesos diferentes.
Random Forest:
- Se comparará el entrenamiento bosques de **100 y 200** arboles de decisión.
- Se variará la máxima profundidad 10 y 20.
Regresión Logística:
-

[93]:
```
# 1. Preprocesamiento
X = datos.drop('NObeyesdad', axis=1)
y = datos['NObeyesdad']
le = LabelEncoder()
y_encoded = le.fit_transform(y)

cat_cols = X.select_dtypes(include='object').columns
num_cols = X.select_dtypes(include=['int64', 'float64']).columns

preprocessor = ColumnTransformer([
    ('num', StandardScaler(), num_cols),
    ('cat', OneHotEncoder(handle_unknown='ignore'), cat_cols)
])
# 2. División
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded,␣
 ↪stratify=y_encoded, test_size=0.3, random_state=42)

# 3. Definir modelos y grids
modelos = {
    'RandomForest': {
        'model': RandomForestClassifier(random_state=42),
        'params': {
            'model__n_estimators': [100, 200],
            'model__max_depth': [None, 10, 20]
        }
    },
    'KNN': {
        'model': KNeighborsClassifier(),
        'params': {
            'model__n_neighbors': [3, 5, 7],
```

```
            'model__weights': ['uniform', 'distance']
        }
    },
    'LogisticRegression': {
        'model': LogisticRegression(max_iter=1000, solver='liblinear'),
        'params': {
            'model__C': [0.1, 1.0, 10.0],
            'model__penalty': ['l1', 'l2']
        }
    }
}

# 4. Validación cruzada
mejores_resultados = {}

cv = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

for nombre, config in modelos.items():
    pipe = Pipeline([
        ('preprocessing', preprocessor),
        ('model', config['model'])
    ])

    grid = GridSearchCV(pipe,
                        param_grid=config['params'],
                        cv=cv,
                        scoring='f1_weighted',
                        n_jobs=-1,
                        verbose=0,
                        return_train_score=False)

    grid.fit(X_train, y_train)

    print(f"\n {nombre}")
    print("Mejores hiperparámetros:", grid.best_params_)
    print(f"Mejor F1 promedio en validación cruzada: {grid.best_score_:.4f}")

    # Evaluación en test set
    y_pred = grid.predict(X_test)
    print("Evaluación en test set:")
    print(classification_report(y_test, y_pred, target_names=le.classes_))

    mejores_resultados[nombre] = {
        'modelo': grid.best_estimator_,
        'f1_cv': grid.best_score_,
        'params': grid.best_params_
    }
```

```python
# 5. Comparar resultados
mejor_modelo = max(mejores_resultados.items(), key=lambda x: x[1]['f1_cv'])
print(f"\n Mejor modelo: {mejor_modelo[0]} con F1 CV =␣
 ↪{mejor_modelo[1]['f1_cv']:.4f}")
```

```
 RandomForest
Mejores hiperparámetros: {'model__max_depth': 20, 'model__n_estimators': 200}
Mejor F1 promedio en validación cruzada: 0.9451
Evaluación en test set:
                     precision    recall  f1-score   support

Insufficient_Weight       1.00      0.93      0.96        82
      Normal_Weight       0.70      0.88      0.78        86
     Obesity_Type_I       0.97      0.93      0.95       106
    Obesity_Type_II       1.00      0.99      0.99        89
   Obesity_Type_III       1.00      0.99      0.99        97
 Overweight_Level_I       0.84      0.82      0.83        87
Overweight_Level_II       0.97      0.89      0.93        87

           accuracy                           0.92       634
          macro avg       0.93      0.92      0.92       634
       weighted avg       0.93      0.92      0.92       634


 KNN
Mejores hiperparámetros: {'model__n_neighbors': 3, 'model__weights': 'distance'}
Mejor F1 promedio en validación cruzada: 0.8379
Evaluación en test set:
                     precision    recall  f1-score   support

Insufficient_Weight       0.84      0.96      0.90        82
      Normal_Weight       0.73      0.41      0.52        86
     Obesity_Type_I       0.89      0.95      0.92       106
    Obesity_Type_II       0.95      0.97      0.96        89
   Obesity_Type_III       0.99      1.00      0.99        97
 Overweight_Level_I       0.69      0.76      0.72        87
Overweight_Level_II       0.74      0.79      0.77        87

           accuracy                           0.84       634
          macro avg       0.83      0.83      0.83       634
       weighted avg       0.84      0.84      0.83       634


 LogisticRegression
Mejores hiperparámetros: {'model__C': 10.0, 'model__penalty': 'l1'}
Mejor F1 promedio en validación cruzada: 0.7860
```

```
Evaluación en test set:
                     precision    recall  f1-score   support

Insufficient_Weight       0.98      1.00      0.99        82
      Normal_Weight       0.71      0.66      0.69        86
     Obesity_Type_I       0.65      0.74      0.69       106
    Obesity_Type_II       0.91      0.96      0.93        89
   Obesity_Type_III       1.00      0.99      0.99        97
 Overweight_Level_I       0.60      0.64      0.62        87
Overweight_Level_II       0.59      0.46      0.52        87

           accuracy                           0.78       634
          macro avg       0.78      0.78      0.78       634
       weighted avg       0.78      0.78      0.78       634


 Mejor modelo: RandomForest con F1 CV = 0.9451
```

[95]:
```
mejor_modelo
```

[95]:
```
('RandomForest',
 {'modelo': Pipeline(steps=[('preprocessing',
                  ColumnTransformer(transformers=[('num', StandardScaler(),
                                                   Index(['Age', 'Height',
'Weight', 'FCVC', 'NCP', 'CH2O', 'FAF', 'TUE'], dtype='object')),
                                                  ('cat',
OneHotEncoder(handle_unknown='ignore'),
                                                   Index(['Gender', 'fwo',
'FAVC', 'CAEC', 'SMOKE', 'SCC', 'CALC', 'MTRANS'], dtype='object'))])),
                 ('model',
                  RandomForestClassifier(max_depth=20, n_estimators=200,
                                         random_state=42))]),
  'f1_cv': 0.9450595430677037,
  'params': {'model__max_depth': 20, 'model__n_estimators': 200}})
```

[96]:
```
!jupyter nbconvert --to pdf --no-input --output Informe.pdf Proyecto.ipynb
```

```
[NbConvertApp] Converting notebook Proyecto.ipynb to html
[NbConvertApp] WARNING | Alternative text is missing on 9 image(s).
[NbConvertApp] Writing 898854 bytes to Informe.html
```