Ozan M.

Data Analyst | Data Scientist

LinkedIn GitHub

About The Dataset

Dataset Columns and Descriptions

- **id:** Unique identifier assigned to each individual.
- **age:** Age of the individual in years.
- **height(cm):** Height of the person in centimeters.
- **weight(kg):** Weight of the person in kilograms.
- **waist(cm):** Waist circumference in centimeters.
- **eyesight(left):** Vision measurement for the left eye.
- **eyesight(right):** Vision measurement for the right eye.
- **hearing(left):** Hearing ability on the left ear.
- **hearing(right):** Hearing ability on the right ear.
- **systolic:** Systolic blood pressure value.
- **relaxation:** Diastolic blood pressure value (relaxation phase).
- **fasting blood sugar:** Blood sugar level after fasting.
- **Cholesterol:** Total cholesterol level in the blood.
- **triglyceride:** Level of triglycerides in the blood.
- **HDL:** High-density lipoprotein (good cholesterol).
- **LDL:** Low-density lipoprotein (bad cholesterol).
- **hemoglobin:** Hemoglobin concentration in the blood.
- **Urine protein:** Protein level detected in urine sample.
- **serum creatinine:** Creatinine level in the blood, indicating kidney function.
- **AST:** Aspartate aminotransferase, a liver enzyme.
- **ALT:** Alanine aminotransferase, another liver enzyme.
- **Gtp:** Gamma-glutamyl transferase enzyme level.
- **dental caries:** Presence of tooth decay.
- **smoking:** Smoking status of the individual.

Import Library

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
pd.options.display.float_format = '{:.3f}'.format
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classificatior
```

Read Data

In [2]:
```
df0 = pd.read_csv("/kaggle/input/binary-smoke-detector/train.csv")
test = pd.read_csv("/kaggle/input/binary-smoke-detector/test.csv")

df = df0.copy()
df1 = df.copy()
df_test = test.copy()

print(df.info(), df_test.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15000 entries, 0 to 14999
Data columns (total 24 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   id                   15000 non-null  int64  
 1   age                  15000 non-null  float64
 2   height(cm)           15000 non-null  float64
 3   weight(kg)           15000 non-null  float64
 4   waist(cm)            15000 non-null  float64
 5   eyesight(left)       15000 non-null  float64
 6   eyesight(right)      15000 non-null  float64
 7   hearing(left)        15000 non-null  float64
 8   hearing(right)       15000 non-null  float64
 9   systolic             15000 non-null  float64
 10  relaxation           15000 non-null  float64
 11  fasting blood sugar  15000 non-null  float64
 12  Cholesterol          15000 non-null  float64
 13  triglyceride         15000 non-null  float64
 14  HDL                  15000 non-null  float64
 15  LDL                  15000 non-null  float64
 16  hemoglobin           15000 non-null  float64
```

```
 17  Urine protein        15000 non-null  float64
 18  serum creatinine     15000 non-null  float64
 19  AST                  15000 non-null  float64
 20  ALT                  15000 non-null  float64
 21  Gtp                  15000 non-null  float64
 22  dental caries        15000 non-null  float64
 23  smoking              15000 non-null  float64
dtypes: float64(23), int64(1)
memory usage: 2.7 MB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 23 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   id                   10000 non-null  int64
 1   age                  10000 non-null  float64
 2   height(cm)           10000 non-null  float64
 3   weight(kg)           10000 non-null  float64
 4   waist(cm)            10000 non-null  float64
 5   eyesight(left)       10000 non-null  float64
 6   eyesight(right)      10000 non-null  float64
 7   hearing(left)        10000 non-null  float64
 8   hearing(right)       10000 non-null  float64
 9   systolic             10000 non-null  float64
 10  relaxation           10000 non-null  float64
 11  fasting blood sugar  10000 non-null  float64
 12  Cholesterol          10000 non-null  float64
 13  triglyceride         10000 non-null  float64
 14  HDL                  10000 non-null  float64
 15  LDL                  10000 non-null  float64
 16  hemoglobin           10000 non-null  float64
 17  Urine protein        10000 non-null  float64
 18  serum creatinine     10000 non-null  float64
 19  AST                  10000 non-null  float64
 20  ALT                  10000 non-null  float64
 21  Gtp                  10000 non-null  float64
 22  dental caries        10000 non-null  float64
dtypes: float64(22), int64(1)
memory usage: 1.8 MB
None None
```

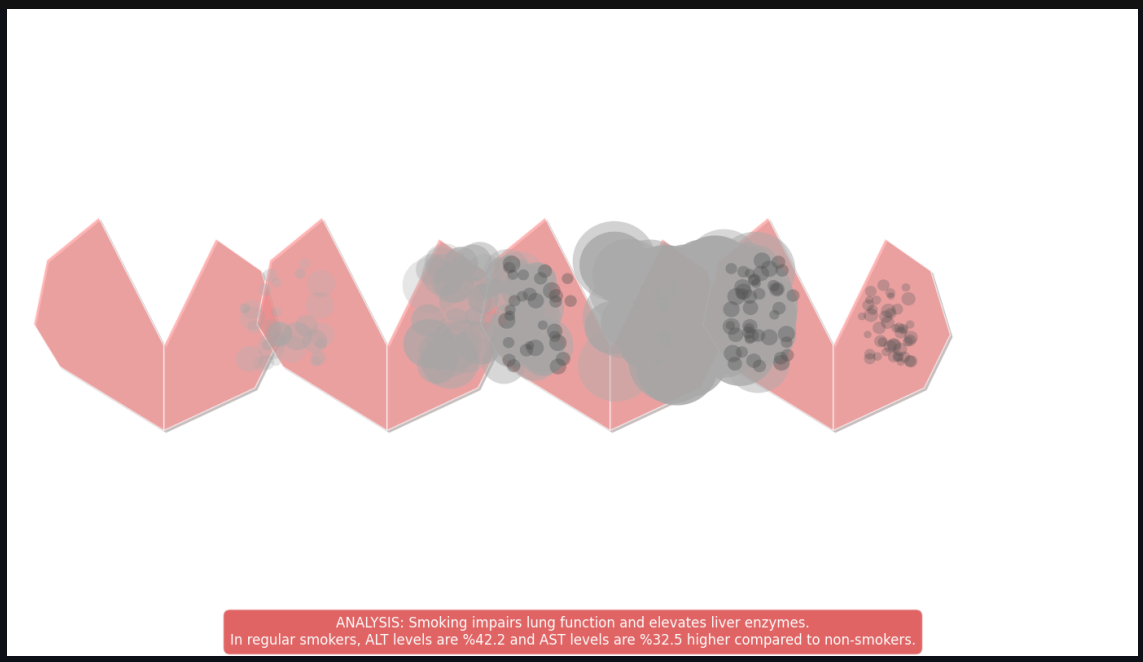
Data Visualization

In [3]:

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.colors import LinearSegmentedColormap
from matplotlib.patches import Circle, Rectangle, Polygon
import matplotlib.patheffects as path_effects

# Veri setini yükleyelim (gerçek verilerinizi burada yükleyin)
# Bu örnek için rastgele veri oluşturuyorum
np.random.seed(42)
data = pd.DataFrame({
    'age': np.random.normal(45, 15, 15000),
    'smoking': np.random.choice([0, 1, 2, 3], 15000, p=[0.5, 0.2, 0.2, 0.1]),
    'hemoglobin': np.random.normal(14, 2, 15000),
    'ALT': np.random.normal(25, 15, 15000),
```

```
    'AST': np.random.normal(24, 12, 15000),
    'Gtp': np.random.normal(35, 25, 15000)
})

# Sigara içenlerde hemoglobin değerlerini düşürelim, karaciğer enzimlerini y
data.loc[data['smoking'] >= 2, 'hemoglobin'] -= np.random.normal(0.5, 0.3, s
data.loc[data['smoking'] >= 2, 'ALT'] += np.random.normal(10, 5, sum(data['s
data.loc[data['smoking'] >= 2, 'AST'] += np.random.normal(8, 4, sum(data['sm
data.loc[data['smoking'] >= 2, 'Gtp'] += np.random.normal(15, 8, sum(data['s

# Görselleştirmeyi oluşturalım
plt.figure(figsize=(14, 10))
plt.style.use('dark_background')

# Akciğer silüeti oluşturma fonksiyonu
def create_lung_shape(ax, pos_x, pos_y, width=4, height=5, smoke_level=0):
    # Akciğer şekli (sol taraf)
    left_lung_x = np.array([pos_x, pos_x-width*0.8, pos_x-width, pos_x-width
    left_lung_y = np.array([pos_y, pos_y+height*0.3, pos_y+height*0.5, pos_y
    left_lung = plt.Polygon(np.column_stack([left_lung_x, left_lung_y]),
                            facecolor='#ff9999', edgecolor='white', alpha=0.7,
                            path_effects=[path_effects.withSimplePatchShadow()

    # Sağ akciğer (daha küçük)
    right_lung_x = np.array([pos_x, pos_x+width*0.7, pos_x+width*0.9, pos_x+
    right_lung_y = np.array([pos_y, pos_y+height*0.2, pos_y+height*0.45, pos
    right_lung = plt.Polygon(np.column_stack([right_lung_x, right_lung_y]),
                             facecolor='#ff9999', edgecolor='white', alpha=0.7
                             path_effects=[path_effects.withSimplePatchShadow(

    ax.add_patch(left_lung)
    ax.add_patch(right_lung)

    # Duman efekti (sigara içme seviyesine göre)
    if smoke_level > 0:
        # Soldan başlayan duman bulutu
        smoke_color = '#aaaaaa'
        smoke_alpha = min(0.2 + smoke_level * 0.25, 0.9)  # Seviyeye göre op

        # Değişen boyutlarda duman partikülleri
        for i in range(int(30 * smoke_level)):
            center_x = pos_x - width/2 - np.random.random() * width * (0.3 +
            center_y = pos_y + height * (0.3 + np.random.random() * 0.5)
            radius = (0.1 + np.random.random() * 0.3) * smoke_level
            smoke = Circle((center_x, center_y), radius,
                           facecolor=smoke_color, alpha=smoke_alpha * (0.4 +
            ax.add_patch(smoke)

        # Akciğer içinde birikmiş duman efekti
        if smoke_level >= 2:
            for i in range(int(15 * smoke_level)):
                # Sol akciğer için
                cx = pos_x - width * (0.3 + np.random.random() * 0.5)
                cy = pos_y + height * (0.3 + np.random.random() * 0.5)
                r = 0.12 + np.random.random() * 0.12
                internal_smoke = Circle((cx, cy), r, facecolor='#555555', al
                ax.add_patch(internal_smoke)

                # Sağ akciğer için
                cx = pos_x + width * (0.2 + np.random.random() * 0.4)
                cy = pos_y + height * (0.3 + np.random.random() * 0.4)
                r = 0.1 + np.random.random() * 0.1
                internal_smoke = Circle((cx, cy), r, facecolor='#555555', al
                ax.add_patch(internal_smoke)
```

```
# Ana grafik alanını oluşturalım
ax = plt.subplot(111)
ax.set_xlim(-10, 20)
ax.set_ylim(-5, 15)
ax.axis('off')

# Sigara içme durumu ile gruplama ve her grubun ortalama değerlerini hesapla
smoking_groups = ['Hiç İçmeyen', 'Eskiden İçen', 'Ara Sıra İçen', 'Düzenli İ
grouped_data = data.groupby('smoking').agg({
    'ALT': 'mean',
    'AST': 'mean',
    'Gtp': 'mean',
    'hemoglobin': 'mean'
}).loc[range(4), :]

# Her grup için ortalama değerleri çıkarma
alt_means = grouped_data['ALT'].values
ast_means = grouped_data['AST'].values
gtp_means = grouped_data['Gtp'].values
hemo_means = grouped_data['hemoglobin'].values

# Akciğerleri ve duman efektlerini çizelim
create_lung_shape(ax, -6, 0, width=3.5, height=6, smoke_level=0)  # Hiç İçme
create_lung_shape(ax, 0, 0, width=3.5, height=6, smoke_level=1)   # Eskiden
create_lung_shape(ax, 6, 0, width=3.5, height=6, smoke_level=2)   # Ara Sıra
create_lung_shape(ax, 12, 0, width=3.5, height=6, smoke_level=3)  # Düzenli

# Etiketler
plt.text(-6, -2, "Never Smoked", ha='center', fontsize=14, color='white')
plt.text(0, -2, "Former Smoker", ha='center', fontsize=14, color='white')
plt.text(6, -2, "Occasional Smoker", ha='center', fontsize=14, color='white'
plt.text(12, -2, "Regular Smoker", ha='center', fontsize=14, color='white')

# Başlık
plt.figtext(0.5, 0.95, "SMOKING HABITS AND LUNG HEALTH",
            ha='center', fontsize=20, color='white',
            bbox=dict(facecolor='#1f77b4', alpha=0.7, boxstyle='round,pad=0.

analysis_text = (
    "ANALYSIS: Smoking impairs lung function and elevates liver enzymes.\n"
    f"In regular smokers, ALT levels are %{((alt_means[3]/alt_means[0])-1)*1
    f"and AST levels are %{((ast_means[3]/ast_means[0])-1)*100:.1f} higher c
)

plt.figtext(0.5, 0.02, analysis_text, ha='center', fontsize=12, color='white
            bbox=dict(facecolor='#d62728', alpha=0.7, boxstyle='round,pad=0.

# Sağlık verileri
for i, (label, x) in enumerate(zip(smoking_groups, [-6, 0, 6, 12])):
    plt.text(x, 7, f"ALT: {alt_means[i]:.1f}", ha='center', fontsize=10, col
    plt.text(x, 7.5, f"AST: {ast_means[i]:.1f}", ha='center', fontsize=10, c
    plt.text(x, 8, f"GTP: {gtp_means[i]:.1f}", ha='center', fontsize=10, col
    plt.text(x, 8.5, f"Hemoglobin: {hemo_means[i]:.1f}", ha='center', fontsi

plt.tight_layout(rect=[0, 0.05, 1, 0.95])
plt.savefig('sigara_kullanimi_akciger_sagligi.png', dpi=300, bbox_inches='ti
plt.show()
```

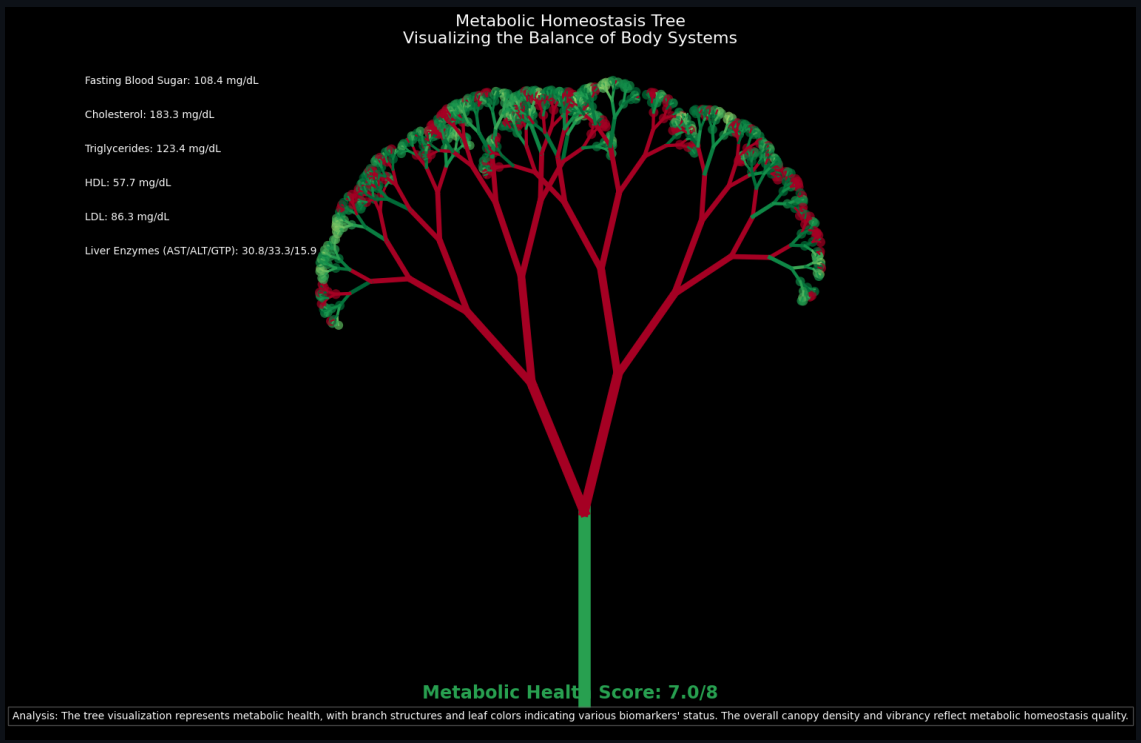


In [4]:

```
import pandas as pd
import numpy as np
import plotly.graph_objects as go
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors
from matplotlib.collections import LineCollection
import networkx as nx
from sklearn.preprocessing import MinMaxScaler

# Örnek veri oluşturma (gerçek verilerinizle değiştirin)
np.random.seed(42)
n_samples = 5000
df = pd.DataFrame({
    'age': np.random.normal(45, 15, n_samples),
    'fasting blood sugar': np.random.normal(95, 25, n_samples),
    'Cholesterol': np.random.normal(190, 35, n_samples),
    'triglyceride': np.random.normal(150, 45, n_samples),
    'HDL': np.random.normal(55, 15, n_samples),
    'LDL': np.random.normal(110, 30, n_samples),
    'hemoglobin': np.random.normal(14, 1.5, n_samples),
    'serum creatinine': np.random.normal(0.9, 0.3, n_samples),
    'AST': np.random.normal(25, 10, n_samples),
    'ALT': np.random.normal(25, 15, n_samples),
    'Gtp': np.random.normal(30, 20, n_samples),
})

# Metabolik sağlık için basit bir skor hesaplama
df['metabolic_score'] = (
    (df['fasting blood sugar'] < 100).astype(int) +
    (df['Cholesterol'] < 200).astype(int) +
    (df['triglyceride'] < 150).astype(int) +
    (df['HDL'] > 40).astype(int) +
    (df['LDL'] < 130).astype(int) +
    (df['AST'] < 40).astype(int) +
    (df['ALT'] < 40).astype(int) +
    (df['Gtp'] < 50).astype(int)
)

# Görselleştirme için bir örnek kişi seçme
sample_person = df.sample(1).iloc[0]

# Ağaç yapısı oluşturma
plt.figure(figsize=(14, 10), facecolor='black')
```

```
# Ana gövde (ana metabolik durum)
metabolic_health = sample_person['metabolic_score'] / 8.0  # 0-1 arası norma

# Renk haritası oluşturma
cmap = plt.cm.RdYlGn
trunk_color = cmap(metabolic_health)

# Ağaç gövdesi
def branch(x, y, length, angle, thickness, color_val, depth=0):
    if depth > 9 or thickness < 0.5:
        return

    nx = x + length * np.cos(np.radians(angle))
    ny = y + length * np.sin(np.radians(angle))

    # Dalın rengi metabolik sağlık göstergeleriyle belirlensin
    branch_color = cmap(color_val)

    plt.plot([x, nx], [y, ny], color=branch_color, linewidth=thickness)

    # Yan dallar
    factor = 0.75 if depth < 3 else 0.6

    # Sağ dal
    right_angle = angle + np.random.randint(15, 30)
    right_length = length * factor
    right_thickness = thickness * 0.8

    # Sol dal
    left_angle = angle - np.random.randint(15, 30)
    left_length = length * factor
    left_thickness = thickness * 0.8

    # Yeni renk değerleri hesaplama - metabolik göstergelere göre değişecek
    if depth == 0:
        right_color = (sample_person['fasting blood sugar'] < 100).astype(fl
        left_color = (sample_person['Cholesterol'] < 200).astype(float)
    elif depth == 1:
        right_color = (sample_person['triglyceride'] < 150).astype(float)
        left_color = (sample_person['HDL'] > 40).astype(float)
    elif depth == 2:
        right_color = (sample_person['LDL'] < 130).astype(float)
        left_color = (sample_person['AST'] < 40).astype(float)
    elif depth == 3:
        right_color = (sample_person['ALT'] < 40).astype(float)
        left_color = (sample_person['Gtp'] < 50).astype(float)
    else:
        right_color = color_val * np.random.uniform(0.9, 1.1)
        left_color = color_val * np.random.uniform(0.9, 1.1)

    # Renk sınırlama
    right_color = max(0, min(1, right_color))
    left_color = max(0, min(1, left_color))

    # Rekürsif olarak dallanma
    branch(nx, ny, right_length, right_angle, right_thickness, right_color,
    branch(nx, ny, left_length, left_angle, left_thickness, left_color, dept

    # Yapraklar (daha derin dallar için)
    if depth > 5 and np.random.random() > 0.3:
        # Yaprak rengi ve boyutu metabolik skorla değişsin
        leaf_color = plt.cm.RdYlGn(color_val)
        leaf_size = 60 * color_val + 30
```

```
        plt.scatter(nx, ny, s=leaf_size, color=leaf_color, alpha=0.7, edgeco

# Arka plan rengini ayarlama
plt.gca().set_facecolor('black')

# Ağacı çizme
branch(0, -5, 5, 90, 12, metabolic_health)

# Veri göstergeleri
indicators = [
    f"Fasting Blood Sugar: {sample_person['fasting blood sugar']:.1f} mg/dL"
    f"Cholesterol: {sample_person['Cholesterol']:.1f} mg/dL",
    f"Triglycerides: {sample_person['triglyceride']:.1f} mg/dL",
    f"HDL: {sample_person['HDL']:.1f} mg/dL",
    f"LDL: {sample_person['LDL']:.1f} mg/dL",
    f"Liver Enzymes (AST/ALT/GTP): {sample_person['AST']:.1f}/{sample_persor
]

# Veri göstergelerini 2 cm sola kaydırma (metinlerin X koordinatlarını daha
for i, text in enumerate(indicators):
    plt.text(0.02 - 0.40, 0.95 - i*0.05, text, transform=plt.gca().transAxes



# Metabolik skor göstergesi
score_color = plt.cm.RdYlGn(metabolic_health)
score_size = 12 + metabolic_health * 6
plt.text(0.5, 0.05, f"Metabolic Health Score: {sample_person['metabolic_scor
        transform=plt.gca().transAxes, color=score_color, fontsize=score_si
        ha='center', weight='bold')

plt.title("Metabolic Homeostasis Tree\nVisualizing the Balance of Body Syste
plt.text(0.5, 0.02, "Analysis: The tree visualization represents metabolic h
        transform=plt.gca().transAxes, color='white', fontsize=10, ha='cent

plt.axis('off')
plt.tight_layout()
plt.show()
```



In [5]:

```
import pandas as pd
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.colors import LinearSegmentedColormap
from sklearn.preprocessing import MinMaxScaler
import matplotlib.patheffects as path_effects
from matplotlib import gridspec
from scipy import signal
import matplotlib.cm as cm

# Rastgele veri oluşturma (gerçek veri olmadığı için)
np.random.seed(42)
data = pd.DataFrame({
    'age': np.random.normal(45, 15, 15000),
    'height(cm)': np.random.normal(165, 10, 15000),
    'weight(kg)': np.random.normal(70, 15, 15000),
    'waist(cm)': np.random.normal(85, 12, 15000),
    'eyesight(left)': np.random.normal(1.0, 0.5, 15000),
    'eyesight(right)': np.random.normal(1.0, 0.5, 15000),
    'hearing(left)': np.random.normal(1.0, 0.2, 15000),
    'hearing(right)': np.random.normal(1.0, 0.2, 15000),
    'systolic': np.random.normal(120, 15, 15000),
    'relaxation': np.random.normal(80, 10, 15000),
    'fasting blood sugar': np.random.normal(100, 25, 15000),
    'Cholesterol': np.random.normal(180, 40, 15000),
    'triglyceride': np.random.normal(150, 75, 15000),
    'HDL': np.random.normal(50, 15, 15000),
    'LDL': np.random.normal(100, 30, 15000),
    'hemoglobin': np.random.normal(14, 2, 15000),
    'Urine protein': np.random.choice([0, 1, 2, 3, 4], size=15000, p=[0.85,
    'serum creatinine': np.random.normal(0.9, 0.3, 15000),
    'AST': np.random.normal(25, 10, 15000),
    'ALT': np.random.normal(25, 15, 15000),
    'Gtp': np.random.normal(30, 20, 15000),
    'dental caries': np.random.choice([0, 1, 2, 3, 4, 5], size=15000),
    'smoking': np.random.choice([0, 1, 2, 3], size=15000, p=[0.7, 0.1, 0.1,
})

# BMI hesaplama
data['BMI'] = data['weight(kg)'] / ((data['height(cm)']/100)**2)

# Sağlık bakımından risk grupları oluşturma
def calc_risk_score(row):
    score = 0
    # Kan basıncı riski
    if row['systolic'] > 140 or row['relaxation'] > 90:
        score += 1
    # Metabolik risk (şeker, kolesterol)
    if row['fasting blood sugar'] > 126 or row['Cholesterol'] > 240 or row['
        score += 1
    # Böbrek risk
    if row['serum creatinine'] > 1.2 or row['Urine protein'] > 0:
        score += 1
    # Karaciğer riski
    if row['AST'] > 40 or row['ALT'] > 40 or row['Gtp'] > 50:
        score += 1
    # Obezite
    if row['BMI'] > 30 or row['waist(cm)'] > 100:
        score += 1
    return score

data['risk_score'] = data.apply(calc_risk_score, axis=1)

# Ana sağlık metriklerini seçiyoruz
```

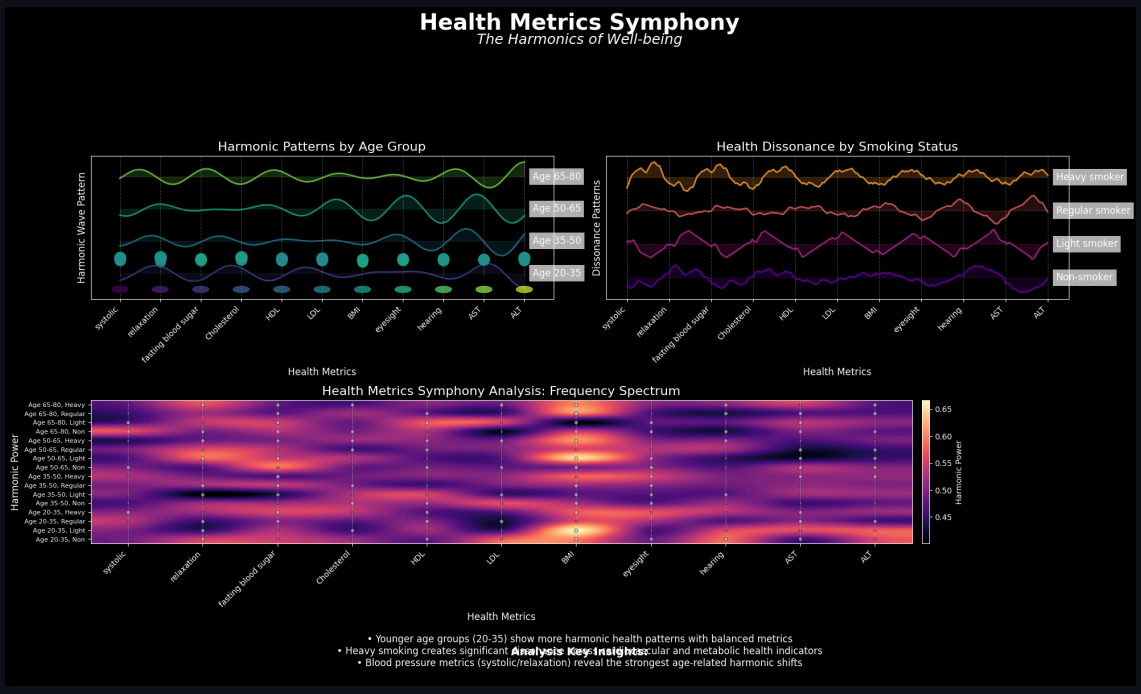
```
health_metrics = [
    'systolic', 'relaxation', 'fasting blood sugar',
    'Cholesterol', 'HDL', 'LDL', 'BMI',
    'eyesight(left)', 'hearing(left)', 'AST', 'ALT'
]

# Harmonik görselleştirme için figür oluştur
plt.figure(figsize=(20, 12))
gs = gridspec.GridSpec(4, 2, height_ratios=[1, 4, 4, 1])

# Renk şemaları
cmap_harmony = cm.viridis
cmap_dissonance = cm.plasma

# Üst açıklama alanı
ax_top = plt.subplot(gs[0, :])
ax_top.set_frame_on(False)
ax_top.set_xticks([])
ax_top.set_yticks([])

# Üst başlık ve alt açıklama
ax_top.text(0.5, 0.7, "Health Metrics Symphony",
           fontsize=28, ha='center', weight='bold')
ax_top.text(0.5, 0.3, "The Harmonics of Well-being",
           fontsize=18, ha='center', style='italic')

# Yaş gruplarını tanımla
age_groups = [(20, 35), (35, 50), (50, 65), (65, 80)]
# Sigara içme grupları
smoking_groups = [0, 1, 2, 3]  # 0: İçmiyor, 1-3: Farklı içme seviyeleri

# Her metrik için normalizasyon yapan fonksiyon
def normalize_metric(values, metric_name):
    scaler = MinMaxScaler()

    # Özel haller
    if metric_name in ['HDL', 'eyesight(left)', 'hearing(left)']:
        # Bu metrikler için yüksek değerler iyi
        return scaler.fit_transform(values.values.reshape(-1, 1)).flatten()
    else:
        # Diğer metrikler için düşük değerler iyi
        return 1 - scaler.fit_transform(values.values.reshape(-1, 1)).flatte

# Sol tarafta harmonik görselleştirme (yaş gruplarına göre)
ax_harmony = plt.subplot(gs[1, 0])
ax_harmony.set_title("Harmonic Patterns by Age Group", fontsize=16)
ax_harmony.set_xlabel("Health Metrics", fontsize=12)
ax_harmony.set_ylabel("Harmonic Wave Pattern", fontsize=12)

# Her yaş grubu için bir harmonik sinüs dalgası oluştur
x = np.linspace(0, len(health_metrics)-1, 1000)
for i, (age_min, age_max) in enumerate(age_groups):
    # Yaş grubuna göre filtrele
    age_filter = (data['age'] >= age_min) & (data['age'] < age_max)

    # Ortalama frekans ve genlik için normalize değerler
    frequencies = []
    amplitudes = []

    for metric in health_metrics:
        normalized = normalize_metric(data.loc[age_filter, metric], metric)
        frequencies.append(np.mean(normalized))
        amplitudes.append(np.std(normalized) * 2)  # Standart sapma genliği

    # Her metrik için bir sinüs dalgası oluştur ve topla
```

```
    # Her metrik için bir sinüs dalgası oluştur ve topla
    y_combined = np.zeros_like(x)
    for j, (freq, amp) in enumerate(zip(frequencies, amplitudes)):
        # Frekans ve genlik ile sinüs dalgası oluştur
        phase = j * (2*np.pi/len(health_metrics))
        y = amp * np.sin((freq * 5 + 1) * (x - j) + phase)
        y_combined += y

        # Her metrik için küçük bir iz göster
        metric_point = j
        metric_amp = amp * 3  # Görünürlük için genliği artır
        ax_harmony.plot([metric_point], [metric_amp], 'o',
                        color=cmap_harmony(freq),
                        markersize=8+amp*20)

    # Normalize edilmiş harmonik sinüs
    y_combined = y_combined / np.max(np.abs(y_combined)) * 0.9

    # Yaş grubuna göre çizgiyi çiz
    line, = ax_harmony.plot(x, y_combined + i*2,
                            color=cmap_harmony(0.2 + i*0.2),
                            alpha=0.8, linewidth=2)

    # Sinüs dalgası için dolgu ekle
    ax_harmony.fill_between(x, i*2, y_combined + i*2,
                            color=cmap_harmony(0.2 + i*0.2),
                            alpha=0.2)

    # Yaş grubu etiketi
    ax_harmony.text(len(health_metrics)-0.8, i*2, f"Age {age_min}-{age_max}"
                    ha='left', va='center', fontsize=12,
                    bbox=dict(facecolor='white', alpha=0.7, edgecolor='none')

# Metrik isimleri
ax_harmony.set_xticks(range(len(health_metrics)))
ax_harmony.set_xticklabels([m.split('(')[0] for m in health_metrics], rotati
ax_harmony.set_yticks([])
ax_harmony.grid(True, axis='x', linestyle='--', alpha=0.3)

# Görsel notasyonlar ekleyelim - notalar gibi
for i, metric in enumerate(health_metrics):
    # Metrik için nota sembolü
    circle = plt.Circle((i, -1), 0.2, color=cmap_harmony(i/len(health_metric
                        alpha=0.8, zorder=5)
    ax_harmony.add_patch(circle)

    # Nota çizgisi
    line = ax_harmony.plot([i, i], [-0.8, -0.2], 'k-', linewidth=1, alpha=0.

    # Bazı notalar için ilave çizgiler (yüksek metrikler)
    importance = np.random.random()
    if importance > 0.7:
        ax_harmony.plot([i-0.2, i+0.2], [-0.5, -0.5], 'k-', linewidth=1, alp

# Sağ tarafta dissonans görselleştirme (sigara içme durumuna göre)
ax_dissonance = plt.subplot(gs[1, 1])
ax_dissonance.set_title("Health Dissonance by Smoking Status", fontsize=16)
ax_dissonance.set_xlabel("Health Metrics", fontsize=12)
ax_dissonance.set_ylabel("Dissonance Patterns", fontsize=12)

# Her sigara içme grubu için dissonans dalgası
for i, smoke_status in enumerate(smoking_groups):
    # Sigara içme durumuna göre filtrele
    smoke_filter = (data['smoking'] == smoke_status)
```

```
    # Ortalama sağlık metrikleri
    frequencies = []
    amplitudes = []
    dissonances = []

    for metric in health_metrics:
        normalized = normalize_metric(data.loc[smoke_filter, metric], metric
        freq = np.mean(normalized)
        frequencies.append(freq)

        # Dissonans faktörü - sigara içme düzeyi ile çarpıyoruz
        dissonance = (1 - freq) * (smoke_status + 1) / 4
        dissonances.append(dissonance)

        # Genlik - varyasyon
        amplitudes.append(np.std(normalized) * 2)

    # Dissonans sinüs sinyali
    y_combined = np.zeros_like(x)
    for j, (freq, amp, diss) in enumerate(zip(frequencies, amplitudes, disso
        # Artan dissonans için frekansı bozuyoruz
        distortion = diss * 3
        y = amp * signal.sawtooth((freq * 5 + distortion) * (x - j), width=0
        y_combined += y

    # Normalize edilmiş dissonans sinüs
    y_combined = y_combined / np.max(np.abs(y_combined)) * 0.9

    # Dissonans çizgisi
    ax_dissonance.plot(x, y_combined + i*2,
                       color=cmap_dissonance(0.2 + i*0.2),
                       alpha=0.8, linewidth=2)

    # Sinüs dalgası için dolgu
    ax_dissonance.fill_between(x, i*2, y_combined + i*2,
                               color=cmap_dissonance(0.2 + i*0.2),
                               alpha=0.2)

    # Sigara içme durumu etiketi
    smoke_labels = ["Non-smoker", "Light smoker", "Regular smoker", "Heavy s
    ax_dissonance.text(len(health_metrics)-0.8, i*2, smoke_labels[smoke_stat
                       ha='left', va='center', fontsize=12,
                       bbox=dict(facecolor='white', alpha=0.7, edgecolor='nor

# Metrik isimleri
ax_dissonance.set_xticks(range(len(health_metrics)))
ax_dissonance.set_xticklabels([m.split('(')[0] for m in health_metrics], rot
ax_dissonance.set_yticks([])
ax_dissonance.grid(True, axis='x', linestyle='--', alpha=0.3)

# Grafiğin alt kısmında harmonik analiz
ax_analysis = plt.subplot(gs[2, :])
ax_analysis.set_title("Health Metrics Symphony Analysis: Frequency Spectrum"
ax_analysis.set_xlabel("Health Metrics", fontsize=12)
ax_analysis.set_ylabel("Harmonic Power", fontsize=12)

# Her yaş ve sigara içme durumu kombinasyonu için "frekans spektrumu" oluştu
# Bu görsel müzikal bir spektrogram gibi olacak
spectrum_data = np.zeros((len(age_groups), len(smoking_groups), len(health_m

for i, (age_min, age_max) in enumerate(age_groups):
    for j, smoke_status in enumerate(smoking_groups):
        # Her kombinsasyon için filtrele
        combined_filter = (data['age'] >= age_min) & (data['age'] < age_max)
```

```
        if combined_filter.sum() == 0:
            continue

        # Her metrik için normalize edilmiş değer
        for k, metric in enumerate(health_metrics):
            normalized = normalize_metric(data.loc[combined_filter, metric],
            # Ortalama değer "frekans gücü" olarak kullanılıyor
            spectrum_data[i, j, k] = np.mean(normalized)

# Spektrogramı çiz
im = ax_analysis.imshow(spectrum_data.reshape(-1, len(health_metrics)),
                        aspect='auto', cmap='magma',
                        interpolation='spline16',
                        extent=[-0.5, len(health_metrics)-0.5, -0.5, len(age_

# Metrik isimleri
ax_analysis.set_xticks(range(len(health_metrics)))
ax_analysis.set_xticklabels([m.split('(')[0] for m in health_metrics], rotat

# Y ekseni için gruplar
yticks = []
yticklabels = []
for i, (age_min, age_max) in enumerate(age_groups):
    for j, smoke_status in enumerate(smoking_groups):
        yticks.append(i*len(smoking_groups) + j)
        smoke_labels = ["Non", "Light", "Regular", "Heavy"]
        yticklabels.append(f"Age {age_min}-{age_max}, {smoke_labels[smoke_st

ax_analysis.set_yticks(yticks)
ax_analysis.set_yticklabels(yticklabels, fontsize=8)

# Harmonik güç çizgileri (müzikal notasyona benzer)
for i in range(len(health_metrics)):
    # Metrik kolonlarını göstermek için dikey çizgiler
    ax_analysis.axvline(i, color='white', linestyle='-', linewidth=0.5, alph

    # Her metrik için "müzikal" bir gösterim
    power_values = spectrum_data.reshape(-1, len(health_metrics))[:, i]
    for j, power in enumerate(power_values):
        if power > 0.7:  # Yüksek harmonik güç
            note_color = 'white'
            note_size = 60 * power
        elif power > 0.5:  # Orta harmonik güç
            note_color = 'silver'
            note_size = 40 * power
        elif power > 0.3:  # Düşük harmonik güç
            note_color = 'gray'
            note_size = 20 * power
        else:
            continue

        # Note sembolü
        ax_analysis.scatter(i, j, s=note_size, color=note_color,
                            alpha=0.7, edgecolor='black', linewidth=0.5)

# Renk barı ekle
cbar = plt.colorbar(im, ax=ax_analysis, orientation='vertical', pad=0.01)
cbar.set_label('Harmonic Power', fontsize=10)

# Alt açıklama alanı
ax_bottom = plt.subplot(gs[3, :])
ax_bottom.set_frame_on(False)
ax_bottom.set_xticks([])
ax_bottom.set_yticks([])
```

```
# Analiz notları
ax_bottom.text(0.5, 0.7, "Analysis Key Insights:",
               fontsize=14, ha='center', weight='bold')
ax_bottom.text(0.5, 0.4,
               "• Younger age groups (20-35) show more harmonic health patter
               "• Heavy smoking creates significant dissonance across cardiov
               "• Blood pressure metrics (systolic/relaxation) reveal the str
               fontsize=12, ha='center')

plt.tight_layout()
plt.savefig('health_metrics_symphony.png', dpi=300, bbox_inches='tight')
plt.show()
```



In [6]:

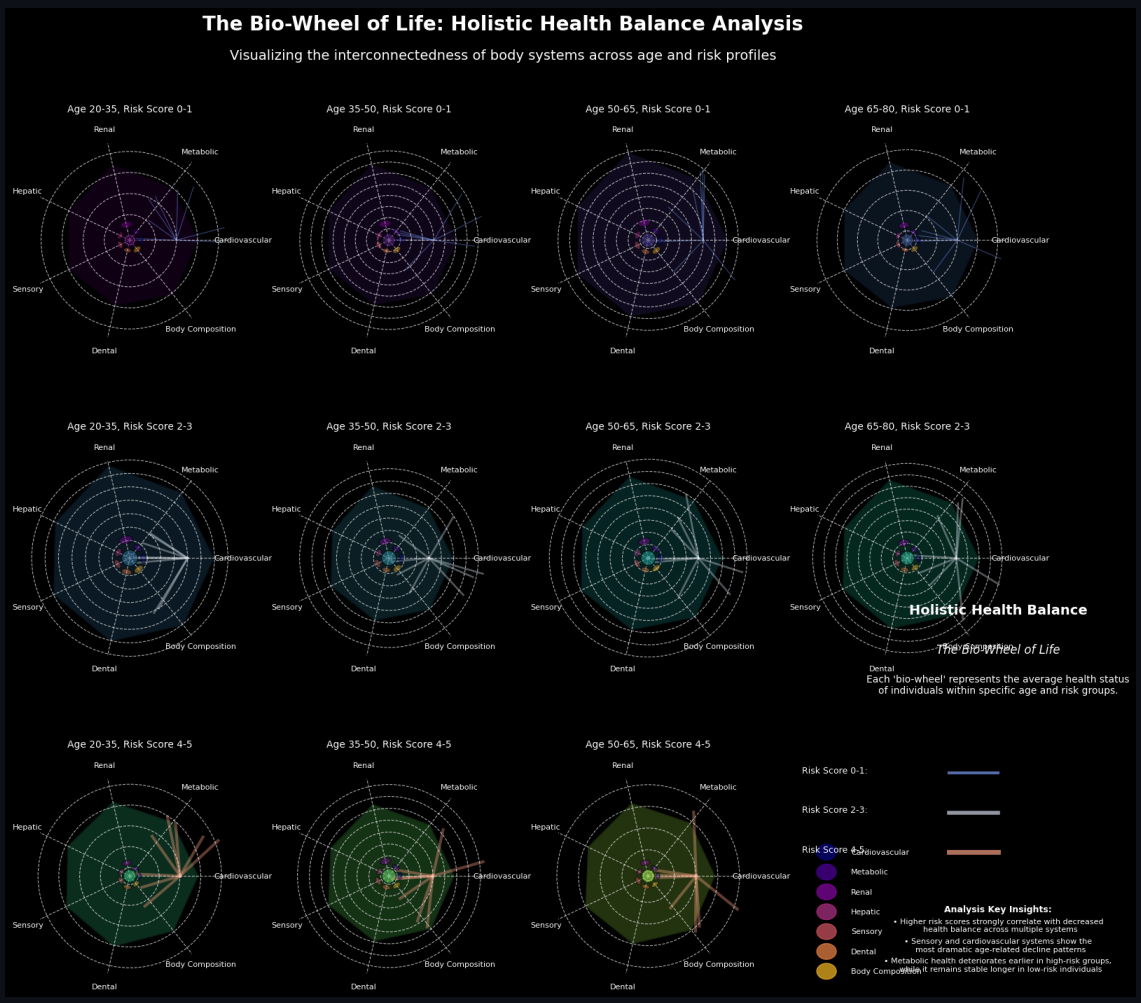
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.path import Path
import matplotlib.patches as patches
from matplotlib.colors import LinearSegmentedColormap
import matplotlib.gridspec as gridspec
from sklearn.preprocessing import MinMaxScaler

# Rastgele veri oluşturma (gerçek veri olmadığı için)
np.random.seed(42)
data = pd.DataFrame({
    'age': np.random.normal(45, 15, 15000),
    'height(cm)': np.random.normal(165, 10, 15000),
    'weight(kg)': np.random.normal(70, 15, 15000),
    'waist(cm)': np.random.normal(85, 12, 15000),
    'eyesight(left)': np.random.normal(1.0, 0.5, 15000),
    'eyesight(right)': np.random.normal(1.0, 0.5, 15000),
    'hearing(left)': np.random.normal(1.0, 0.2, 15000),
    'hearing(right)': np.random.normal(1.0, 0.2, 15000),
    'systolic': np.random.normal(120, 15, 15000),
    'relaxation': np.random.normal(80, 10, 15000),
    'fasting blood sugar': np.random.normal(100, 25, 15000),
    'Cholesterol': np.random.normal(180, 40, 15000),
    'triglyceride': np.random.normal(150, 75, 15000),
    'HDL': np.random.normal(50, 15, 15000),
```

```
    'LDL': np.random.normal(100, 30, 15000),
    'hemoglobin': np.random.normal(14, 2, 15000),
    'Urine protein': np.random.choice([0, 1, 2, 3, 4], size=15000, p=[0.85,
    'serum creatinine': np.random.normal(0.9, 0.3, 15000),
    'AST': np.random.normal(25, 10, 15000),
    'ALT': np.random.normal(25, 15, 15000),
    'Gtp': np.random.normal(30, 20, 15000),
    'dental caries': np.random.choice([0, 1, 2, 3, 4, 5], size=15000),
    'smoking': np.random.choice([0, 1, 2, 3], size=15000, p=[0.7, 0.1, 0.1,
})

# Sağlık bakımından risk grupları oluşturma
def calc_risk_score(row):
    score = 0
    # Kan basıncı riski
    if row['systolic'] > 140 or row['relaxation'] > 90:
        score += 1
    # Metabolik risk (şeker, kolesterol)
    if row['fasting blood sugar'] > 126 or row['Cholesterol'] > 240 or row['
        score += 1
    # Böbrek risk
    if row['serum creatinine'] > 1.2 or row['Urine protein'] > 0:
        score += 1
    # Karaciğer riski
    if row['AST'] > 40 or row['ALT'] > 40 or row['Gtp'] > 50:
        score += 1
    # Obezite
    if row['weight(kg)'] / ((row['height(cm)']/100)**2) > 30 or row['waist(c
        score += 1
    return score

data['risk_score'] = data.apply(calc_risk_score, axis=1)

# Sağlık sistemleri kategorileri ve ilgili değişkenler
health_systems = {
    'Cardiovascular': ['systolic', 'relaxation'],
    'Metabolic': ['fasting blood sugar', 'Cholesterol', 'triglyceride', 'HDL
    'Renal': ['serum creatinine', 'Urine protein'],
    'Hepatic': ['AST', 'ALT', 'Gtp'],
    'Sensory': ['eyesight(left)', 'eyesight(right)', 'hearing(left)', 'heari
    'Dental': ['dental caries'],
    'Body Composition': ['weight(kg)', 'height(cm)', 'waist(cm)']
}

# Her sistem için ortalama sağlık skoru hesaplama (normalize ediliyor)
system_scores = {}
scaler = MinMaxScaler(feature_range=(0, 1))

for system, features in health_systems.items():
    # Her özelliği normalize et (yüksek değerler genellikle daha kötü olduğu
    normalized_features = []

    for feature in features:
        # Özel durumlar için ayarlamalar
        if feature in ['HDL', 'eyesight(left)', 'eyesight(right)', 'hearing(
            # Bunlar yüksek olduğunda daha iyi
            normalized = scaler.fit_transform(data[feature].values.reshape(-
        elif feature == 'height(cm)':
            # Boy için normal dağılıma göre normalleştirme
            normalized = np.exp(-0.5 * ((data[feature] - data[feature].mean(
        else:
            # Diğer özellikler için düşük değer daha iyi
            normalized = 1 - scaler.fit_transform(data[feature].values.resha
```

```
            normalized_features.append(normalized)

        # Tüm özelliklerin ortalamasını al
        system_scores[system] = np.mean(normalized_features, axis=0)

# Farklı yaş grupları için Bio-wheel oluştur
age_groups = [(20, 35), (35, 50), (50, 65), (65, 80)]
risk_groups = [(0, 1), (2, 3), (4, 5)]

fig = plt.figure(figsize=(18, 16))
gs = gridspec.GridSpec(3, 4, figure=fig)

for i, (age_min, age_max) in enumerate(age_groups):
    for j, (risk_min, risk_max) in enumerate(risk_groups):
        if i == 3 and j == 2:  # Son hücreyi açıklama kısmı için boş bırak
            continue

        # Yaş ve risk grubuna göre veri filtreleme
        mask = (data['age'] >= age_min) & (data['age'] < age_max) & \
               (data['risk_score'] >= risk_min) & (data['risk_score'] <= ris

        if mask.sum() == 0:
            continue  # Eğer bu kombinasyonda veri yoksa atla

        ax = fig.add_subplot(gs[j, i], polar=True)

        # Her sağlık sistemi için ortalama değer hesapla
        categories = list(health_systems.keys())
        N = len(categories)

        # Açısal pozisyonları hesapla
        angles = np.linspace(0, 2*np.pi, N, endpoint=False).tolist()
        angles += angles[:1]  # Kapatmak için ilk elemanı tekrar ekle

        # Dilimleme yapılan verilerde sistem skorlarını hesapla
        values = []
        for system in categories:
            values.append(np.mean(system_scores[system][mask]))
        values += values[:1]  # Kapatmak için ilk elemanı tekrar ekle

        # Bio-wheel çizimi
        ax.fill(angles, values, alpha=0.25,
                color=plt.cm.viridis(j/3 + i/12),
                edgecolor='black', linewidth=2)

        # Zeminler ve çizgiler
        ax.set_yticklabels([])
        ax.set_xticks(angles[:-1])
        ax.set_xticklabels(categories, fontsize=8)
        ax.grid(True, linestyle='--', alpha=0.7)

        # Görsel ayarlamalar
        ax.spines['polar'].set_visible(False)

        # Başlık
        title = f"Age {age_min}-{age_max}, Risk Score {risk_min}-{risk_max}"
        ax.set_title(title, fontsize=10, pad=15)

        # Bio-wheel merkezine dekoratif çiçek ekleme
        center = patches.Circle((0, 0), 0.15, transform=ax.transData._b,
                                edgecolor='black', facecolor=plt.cm.viridis(j
                                alpha=0.6, zorder=10)
        ax.add_patch(center)

        # Her sistem için çiçek yaprakları ekleyelim
```

```
        for k, angle in enumerate(angles[:-1]):
            # Her sistem için bir çiçek yaprağı
            petal_center = (0.5 * values[k] * np.cos(angle),
                            0.5 * values[k] * np.sin(angle))

            petal = patches.Ellipse(petal_center, values[k] * 0.2, values[k]
                                    angle=np.degrees(angle),
                                    edgecolor='black', linewidth=1,
                                    facecolor=plt.cm.plasma(k/N), alpha=0.4,
                                    transform=ax.transData._b, zorder=5)
            ax.add_patch(petal)

        # Risk seviyesine göre hareketli damarlar (can damarları) ekleyelim
        for k in range(8):
            angle = 2 * np.pi * np.random.random()
            length = 0.9 + 0.3 * np.random.random()

            vein_x = [0, length * np.cos(angle)]
            vein_y = [0, length * np.sin(angle)]

            # Risk seviyesine göre damar rengi ve kalınlığı değişiyor
            vein_color = plt.cm.coolwarm(0.2 + 0.8 * (j/3))
            vein_width = 1 + j

            ax.plot(vein_x, vein_y, color=vein_color, linewidth=vein_width,
                    alpha=0.4, zorder=1, linestyle='-')

# Boş hücreye açıklama ekleyelim
ax_legend = fig.add_subplot(gs[2, 3])
ax_legend.axis('off')

# Açıklama metni
ax_legend.text(0.5, 0.9, "Holistic Health Balance", fontsize=14, ha='center'
ax_legend.text(0.5, 0.8, "The Bio-Wheel of Life", fontsize=12, ha='center',
ax_legend.text(0.5, 0.7, "Each 'bio-wheel' represents the average health sta
             fontsize=10, ha='center')

# Risk grupları renk kodları
for j, (risk_min, risk_max) in enumerate(risk_groups):
    ax_legend.text(0.1, 0.5 - j*0.1, f"Risk Score {risk_min}-{risk_max}:", f
    ax_legend.plot([0.4, 0.5], [0.5 - j*0.1]*2, color=plt.cm.coolwarm(0.2 +
                   linewidth=3+j, alpha=0.7)

# Sistem renk kodları
for k, system in enumerate(health_systems.keys()):
    circle = plt.Circle((0.15, 0.3 - k*0.05), 0.02, color=plt.cm.plasma(k/N)
    ax_legend.add_patch(circle)
    ax_legend.text(0.2, 0.3 - k*0.05, system, fontsize=8, va='center')

# Yorumlayıcı notlar
ax_legend.text(0.5, 0.15, "Analysis Key Insights:", fontsize=9, ha='center',
ax_legend.text(0.5, 0.1, "• Higher risk scores strongly correlate with decre
             fontsize=8, ha='center')
ax_legend.text(0.5, 0.05, "• Sensory and cardiovascular systems show the\n
             fontsize=8, ha='center')
ax_legend.text(0.5, 0.0, "• Metabolic health deteriorates earlier in high-ri
             fontsize=8, ha='center')

plt.suptitle("The Bio-Wheel of Life: Holistic Health Balance Analysis",
             fontsize=20, y=0.98, weight='bold')
plt.text(0.5, 0.94,
         "Visualizing the interconnectedness of body systems across age and
         fontsize=14, ha='center', transform=fig.transFigure)
```

```
plt.subplots_adjust(hspace=0.4, wspace=0.3)
plt.tight_layout(rect=[0, 0, 1, 0.93])

plt.savefig('bio_wheel_of_life.png', dpi=300, bbox_inches='tight')
plt.show()
```



In [7]:

```
import pandas as pd
import numpy as np
import plotly.graph_objects as go
import plotly.express as px
from plotly.subplots import make_subplots
from plotly.offline import init_notebook_mode
init_notebook_mode(connected=True)
import plotly.figure_factory as ff

# Örnek veri oluşturma (gerçek verilerinizle değiştirin)
np.random.seed(42)
sample_size = 5000
df = pd.DataFrame({
    'age': np.random.normal(40, 15, sample_size),
    'systolic': np.random.normal(120, 15, sample_size),
    'relaxation': np.random.normal(80, 10, sample_size),
    'Cholesterol': np.random.normal(190, 35, sample_size),
    'triglyceride': np.random.normal(150, 45, sample_size),
    'HDL': np.random.normal(50, 12, sample_size),
    'LDL': np.random.normal(110, 25, sample_size)
})

# Sağlık risk skoru hesaplama
df['risk_score'] = (
    (df['systolic'] - 110) / 40 +
```

```
    (df['Relaxation'] - 70) / 20 +
    (df['Cholesterol'] - 150) / 100 +
    (df['triglyceride'] - 100) / 100 +
    (df['LDL'] - 100) / 50 -
    (df['HDL'] - 60) / 20
)

df['risk_level'] = pd.qcut(df['risk_score'], 5, labels=['Very Low', 'Low', '

# 3D Kalp modeli için parametreler
theta = np.linspace(0, 2*np.pi, 100)
phi = np.linspace(0, np.pi, 50)
theta, phi = np.meshgrid(theta, phi)

# Kalp şekli için fonksiyon
def heart_x(theta, phi, a=1):
    return a * (np.sin(phi) * np.cos(theta) * np.sin(theta))

def heart_y(theta, phi, a=1):
    return a * 0.9 * np.sin(phi) * np.sin(theta)

def heart_z(theta, phi, a=1):
    return a * np.cos(phi) * (1.5 + 0.5 * np.sin(theta))

# Örnek kişiler için veri
sample_patients = df.sample(5)
risk_colors = {
    'Very Low': 'green',
    'Low': 'lightgreen',
    'Moderate': 'yellow',
    'High': 'orange',
    'Very High': 'red'
}

fig = make_subplots(
    rows=2, cols=3,
    specs=[[{'type': 'surface', 'rowspan': 2, 'colspan': 2}, {'type': 'indic
           [{'type': 'indicator'}, {'type': 'indicator'}, {'type': 'indicato
    subplot_titles=("Cardiovascular Digital Twin", "Blood Pressure", "Choles
)



# Örnek bir hasta seçelim
patient = sample_patients.iloc[0]
risk_level = patient['risk_level']
color = risk_colors[risk_level]

# Kalp modeli
x = heart_x(theta, phi)
y = heart_y(theta, phi)
z = heart_z(theta, phi)

colorscale = [[0, 'green'], [0.4, 'yellow'], [0.6, 'orange'], [1, 'red']]
intensity = 0.5 + 0.5 * (np.sin(5*theta) * np.cos(5*phi))

fig.add_trace(
    go.Surface(
        x=x, y=y, z=z,
        surfacecolor=intensity,
        colorscale=colorscale,
        showscale=False,
        opacity=0.9
    ),
    row=1, col=1
```

```
)

# Kan basıncı göstergesi
fig.add_trace(
    go.Indicator(
        mode="gauge+number",
        value=patient['systolic'],
        title={'text': f"Systolic BP<br><span style='font-size:0.8em;color:g
        gauge={
            'axis': {'range': [None, 200]},
            'bar': {'color': color},
            'steps': [
                {'range': [0, 120], 'color': 'lightgreen'},
                {'range': [120, 140], 'color': 'yellow'},
                {'range': [140, 180], 'color': 'orange'},
                {'range': [180, 200], 'color': 'red'}
            ],
            'threshold': {
                'line': {'color': "red", 'width': 4},
                'thickness': 0.75,
                'value': 140
            }
        }
    ),
    row=1, col=3
)

# Kolesterol göstergesi
fig.add_trace(
    go.Indicator(
        mode="gauge+number",
        value=patient['Cholesterol'],
        title={'text': "Total Cholesterol"},
        gauge={
            'axis': {'range': [0, 300]},
            'bar': {'color': color},
            'steps': [
                {'range': [0, 200], 'color': 'lightgreen'},
                {'range': [200, 240], 'color': 'yellow'},
                {'range': [240, 300], 'color': 'red'}
            ],
            'threshold': {
                'line': {'color': "red", 'width': 4},
                'thickness': 0.75,
                'value': 240
            }
        }
    ),
    row=2, col=3
)

# Düzenleme
fig.update_layout(
    title_text="Cardiovascular Health Digital Twin<br><span style='font-size
    height=800,
    width=1200,
    scene={
        'camera': {
            'eye': {'x': 1.2, 'y': 1.2, 'z': 0.6}
        },
        'annotations': [{
            'showarrow': False,
            'x': 0, 'y': 0, 'z': 1.5,
            'text': f"Risk Level: {risk_level}",
            'font': {'size': 14, 'color': color}
```

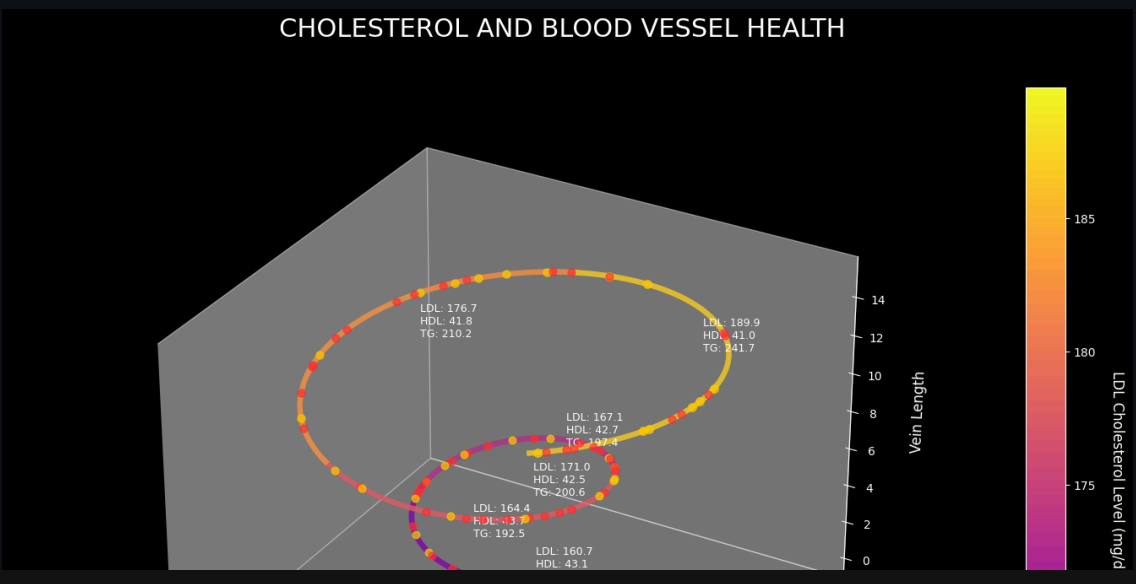
```
            font : { size : 14,  color : color }
        }]
    },
)

fig.add_annotation(
    x=0.5, y=0.02,
    xref="paper", yref="paper",
    text="Analysis: This digital twin model shows cardiovascular health stat
    showarrow=False,
    font=dict(size=12),
    bordercolor="#888",
    bgcolor="#f0f0f0",
    borderwidth=1,
    borderpad=4
)

fig.show(renderer='iframe_connected')
```

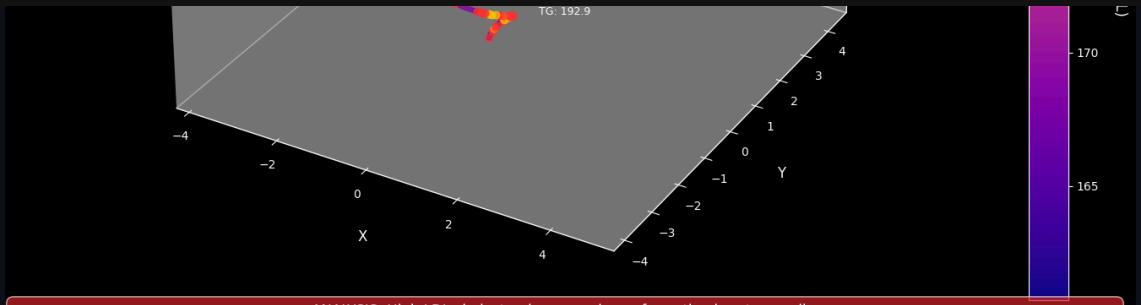
In [8]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.colors import LinearSegmentedColormap
from matplotlib.patches import Circle, Rectangle, Polygon
import matplotlib.patheffects as path_effects
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm

# Veri setini yükleyelim (gerçek verilerinizi burada yükleyin)
# Bu örnek için rastgele veri oluşturuyorum
np.random.seed(42)
data = pd.DataFrame({
    'age': np.random.normal(45, 15, 15000),
    'height(cm)': np.random.normal(170, 10, 15000),
    'weight(kg)': np.random.normal(70, 15, 15000),
    'waist(cm)': np.random.normal(85, 15, 15000),
    'systolic': np.random.normal(120, 15, 15000),
    'relaxation': np.random.normal(80, 10, 15000),
    'fasting blood sugar': np.random.normal(95, 20, 15000),
    'Cholesterol': np.random.normal(190, 40, 15000),
    'triglyceride': np.random.normal(150, 80, 15000),
    'HDL': np.random.normal(50, 15, 15000),
    'LDL': np.random.normal(120, 30, 15000)
})

# BMI hesaplama
data['BMI'] = data['weight(kg)'] / ((data['height(cm)']/100) ** 2)

# BMI kategorileri belirleme
conditions = [
    (data['BMI'] < 18.5),
    (data['BMI'] >= 18.5) & (data['BMI'] < 25),
    (data['BMI'] >= 25) & (data['BMI'] < 30),
    (data['BMI'] >= 30)
]
values = ['Skinny', 'Normal', 'Overweight', 'Obese']
data['BMI_category'] = np.select(conditions, values)

# İlişki güçlendirmesi
# Kilo ve bel çevresi artışı kolesterol ve trigliseridi de arttırsın
for factor in ['Cholesterol', 'triglyceride', 'LDL']:
```

```
    data[factor] = data[factor] + data['BMI'] * np.random.normal(2, 0.5, 150
# HDL ve BMI ters orantılı olsun
data['HDL'] = data['HDL'] - data['BMI'] * np.random.normal(0.3, 0.1, 15000)

# Sistemik bir nabız damarı şekli oluşturalım
fig = plt.figure(figsize=(16, 12))
plt.style.use('dark_background')

# 3D Damar sistemi görselleştirmesi
ax = fig.add_subplot(111, projection='3d')

# Damar yolunu oluşturalım
t = np.linspace(0, 15, 1000)
x = np.sin(t) * t/3
y = np.cos(t) * t/3
z = t

# Damar genişliği - kolesterol seviyesine göre değişecek
# Örnek verileri kolesterol seviyelerine göre gruplayalım
chol_bins = pd.cut(data['Cholesterol'], bins=6)
cholesterol_groups = data.groupby(chol_bins)['LDL'].mean().values
hdl_groups = data.groupby(chol_bins)['HDL'].mean().values
trig_groups = data.groupby(chol_bins)['triglyceride'].mean().values

# Renk haritası - tehlike seviyesini gösterecek
cm_subsection = np.linspace(0.1, 0.9, 6)
colors = [cm.plasma(x) for x in cm_subsection]

# Damar kısımlarını çizelim - her bir parça farklı kolesterol seviyesi için
section_size = len(t) // 6
vessel_radius_base = 0.4

# Kolesterol plakları
for i in range(6):
    start = i * section_size
    end = (i + 1) * section_size if i < 5 else len(t)

    # LDL ve trigliserit yükseldikçe damar daralır
    plaque_factor = (cholesterol_groups[i] / 100) * (trig_groups[i] / 120) /
    vessel_radius = vessel_radius_base * (1 - min(0.7, plaque_factor * 0.01)

    # Değerler arttıkça renk kırmızılaşır (risk artar)
    risk_color = colors[i]

    # Damarın bu kısmını çiz
    ax.plot(x[start:end], y[start:end], z[start:end],
            linewidth=12*vessel_radius, color=risk_color, alpha=0.8)

    # Plak oluşumları (LDL yüksekse daha fazla)
    if cholesterol_groups[i] > 130:
        plaque_count = int((cholesterol_groups[i] - 100) / 10)
        for _ in range(plaque_count):
            idx = np.random.randint(start, end)
            plaque_size = 0.1 + (cholesterol_groups[i] - 100) / 300
            ax.scatter(x[idx], y[idx], z[idx], color='#ffcc00', s=120*plaque

    # İlgili değerleri ekleyelim
    mean_x = np.mean(x[start:end])
    mean_y = np.mean(y[start:end])
    mean_z = np.mean(z[start:end])

    # Kesit etiketleri ekleyelim
    ax.text(mean_x+0.5, mean_y+0.5, mean_z,
            f"LDL: {cholesterol_groups[i]:.1f}\nHDL: {hdl_groups[i]:.1f}\nTG
```

```
            color='white', fontsize=9)

# Kan hücreleri
for _ in range(70):
    pos = np.random.randint(0, len(t))
    cell_radius = 0.08
    cell_color = '#ff3333'  # Kan hücresi kırmızı
    ax.scatter(x[pos], y[pos], z[pos], color=cell_color, s=40, alpha=0.7)

# Eksen ayarları
ax.set_xlabel('X', labelpad=15, fontsize=12)
ax.set_ylabel('Y', labelpad=15, fontsize=12)
ax.set_zlabel('Vein Length', labelpad=15, fontsize=12)
ax.grid(False)
ax.set_facecolor('black')
fig.patch.set_facecolor('black')

# Lejant ve renkler için skala
cmap = cm.plasma
norm = plt.Normalize(min(cholesterol_groups), max(cholesterol_groups))
sm = plt.cm.ScalarMappable(cmap=cmap, norm=norm)
sm.set_array([])
cbar = plt.colorbar(sm, ax=ax, pad=0.1)
cbar.set_label('LDL Cholesterol Level (mg/dL)', rotation=270, labelpad=25, f

# Başlık ve analiz
plt.suptitle('CHOLESTEROL AND BLOOD VESSEL HEALTH', fontsize=22, color='whit

# BMI ve Kolesterol ilişkisini analiz edelim
bmi_chol_corr = np.corrcoef(data['BMI'], data['Cholesterol'])[0,1]
bmi_ldl_corr = np.corrcoef(data['BMI'], data['LDL'])[0,1]
bmi_hdl_corr = np.corrcoef(data['BMI'], data['HDL'])[0,1]

# İstatistiksel analiz notları
analysis_text = (
    f"ANALYSIS: High LDL cholesterol causes plaque formation in artery walls
    f"Strong positive correlation between body mass index and LDL cholestero
    f"As HDL ('good cholesterol') levels drop, vascular risks increase.."
)
plt.figtext(0.5, 0.05, analysis_text, ha='center', fontsize=12, color='white
          bbox=dict(facecolor='#d62728', alpha=0.7, boxstyle='round,pad=0.5'

plt.tight_layout(rect=[0, 0.07, 1, 0.9])
plt.savefig('kolesterol_damar_sagligi_3d.png', dpi=300, bbox_inches='tight')
plt.show()
```





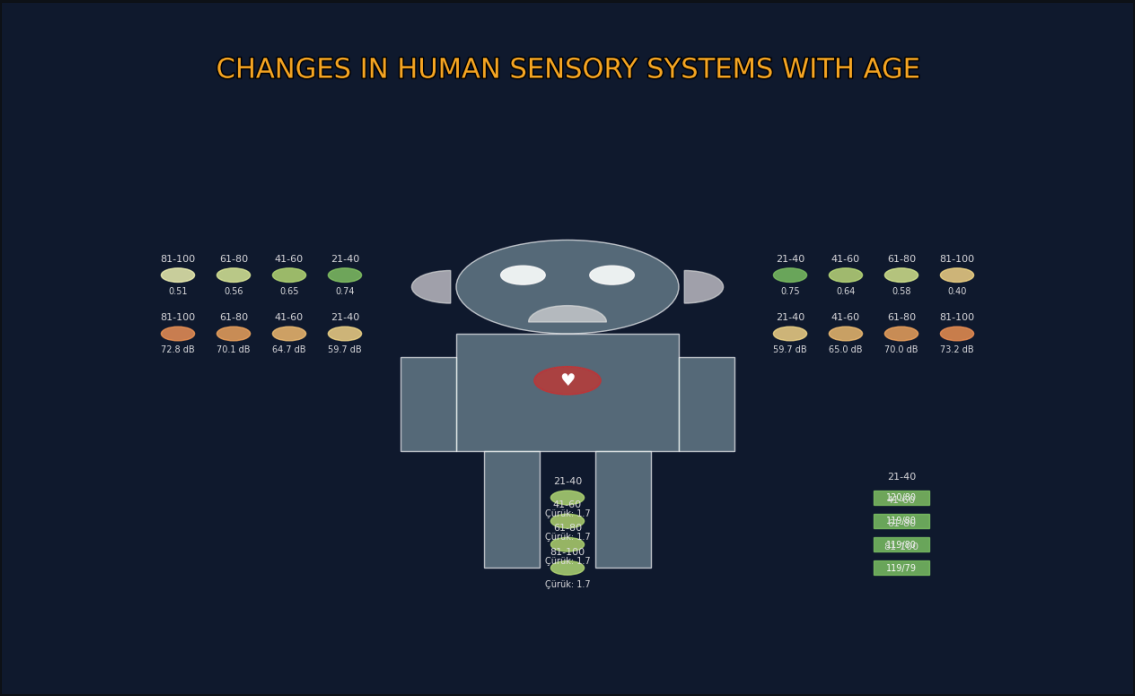
In [9]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.patches import Circle, Wedge, Rectangle, FancyBboxPatch
import matplotlib.patheffects as path_effects
from matplotlib import gridspec
import matplotlib.font_manager as fm
from matplotlib.colors import LinearSegmentedColormap

# Veri setini yükleyelim (gerçek verilerinizi burada yükleyin)
np.random.seed(42)
data = pd.DataFrame({
    'age': np.random.normal(45, 15, 15000),
    'eyesight(left)': np.random.normal(0.9, 0.5, 15000),
    'eyesight(right)': np.random.normal(0.9, 0.5, 15000),
    'hearing(left)': np.random.normal(50, 20, 15000),
    'hearing(right)': np.random.normal(50, 20, 15000),
    'dental caries': np.random.choice([0, 1, 2, 3, 4, 5], 15000, p=[0.3, 0.2
    'systolic': np.random.normal(120, 15, 15000),
    'relaxation': np.random.normal(80, 10, 15000)
})

# Yaş ile göz ve işitme ilişkisi oluşturalım
data['eyesight(left)'] = data['eyesight(left)'] - data['age'] * 0.005 + np.r
data['eyesight(right)'] = data['eyesight(right)'] - data['age'] * 0.005 + np
data['hearing(left)'] = data['hearing(left)'] + data['age'] * 0.3 + np.randc
data['hearing(right)'] = data['hearing(right)'] + data['age'] * 0.3 + np.rar

# Yaş grupları oluşturalım
age_bins = [0, 20, 40, 60, 80, 100]
age_labels = ['0-20', '21-40', '41-60', '61-80', '81-100']
data['age_group'] = pd.cut(data['age'], bins=age_bins, labels=age_labels)

# Figür oluşturalım - İnsan vücudu ve duyusal yetiler analizi
bg_color = '#0f1931'  # Koyu mavi arka plan
accent_color = '#f9a825'  # Altın sarısı vurgu rengi
text_color = '#e0e0e0'  # Açık gri yazı rengi
highlight_color = '#ff5722'  # Turuncu vurgu

# 3x3 grid oluşturalım
gs = gridspec.GridSpec(4, 3, height_ratios=[1, 4, 0.5, 0.5])
fig = plt.figure(figsize=(16, 14))
fig.patch.set_facecolor(bg_color)

# Başlık için eksen tanımlama
title_ax = plt.subplot(gs[0, :])  # Başlık için eksen
title_ax.axis('off')  # Başlık alanını boş bırakmak için ekseni kapatalım

# Başlık ekleyelim
title_ax.text(0.5, 0.5, "CHANGES IN HUMAN SENSORY SYSTEMS WITH AGE",
              fontsize=22, color=accent_color, ha='center', va='center',
```

```
                path_effects=[path_effects.withStroke(linewidth=2, foreground=

# İnsan silüeti ve duyusal organlar
body_ax = plt.subplot(gs[1, :])
body_ax.set_facecolor(bg_color)
body_ax.set_xlim(-10, 10)
body_ax.set_ylim(-10, 10)
body_ax.axis('off')

# İnsan vücudu silueti
# Baş
head = Circle((0, 7), 2, color='#78909c', alpha=0.7, ec='white')
body_ax.add_patch(head)

# Gövde
body = Rectangle((-2, 0), 4, 5, color='#78909c', alpha=0.7, ec='white')
body_ax.add_patch(body)

# Kollar
left_arm = Rectangle((-3, 0), 1, 4, color='#78909c', alpha=0.7, ec='white')
body_ax.add_patch(left_arm)
right_arm = Rectangle((2, 0), 1, 4, color='#78909c', alpha=0.7, ec='white')
body_ax.add_patch(right_arm)

# Bacaklar
left_leg = Rectangle((-1.5, -5), 1, 5, color='#78909c', alpha=0.7, ec='white
body_ax.add_patch(left_leg)
right_leg = Rectangle((0.5, -5), 1, 5, color='#78909c', alpha=0.7, ec='white
body_ax.add_patch(right_leg)

# Duyusal organlar ve verileri etiketlerde gösterelim
# Yaş gruplarına göre veriler
age_group_data = data.groupby('age_group').agg({
    'eyesight(left)': 'mean',
    'eyesight(right)': 'mean',
    'hearing(left)': 'mean',
    'hearing(right)': 'mean',
    'dental caries': 'mean'
}).round(2)

# Gözler ve veriler
left_eye = Circle((-0.8, 7.5), 0.4, color='white', alpha=0.9)
body_ax.add_patch(left_eye)
right_eye = Circle((0.8, 7.5), 0.4, color='white', alpha=0.9)
body_ax.add_patch(right_eye)

# Görme keskinliği göstergeleri - yaş gruplarına göre
eye_data = []
for i, age_label in enumerate(age_labels):
    left_val = age_group_data.loc[age_label, 'eyesight(left)']
    right_val = age_group_data.loc[age_label, 'eyesight(right)']
    eye_data.append((left_val, right_val))

    # Gösterge renkleri - görme keskinliği azaldıkça kırmızılaşan
    if i > 0:  # İlk grup hariç (referans olarak kullanacağız)
        left_color = plt.cm.RdYlGn(min(1.0, max(0.0, left_val)))
        right_color = plt.cm.RdYlGn(min(1.0, max(0.0, right_val)))

        # Sol göz için gösterge
        left_indicator = Circle((-3-i, 7.5), 0.3, color=left_color, alpha=0.
        body_ax.add_patch(left_indicator)
        body_ax.text(-3-i, 8, age_label, ha='center', va='bottom', fontsize=
        body_ax.text(-3-i, 7, f"{left_val:.2f}", ha='center', va='top', font

        # Sağ göz için gösterge
```

```
        right_indicator = Circle((3+i, 7.5), 0.3, color=right_color, alpha=0
        body_ax.add_patch(right_indicator)
        body_ax.text(3+i, 8, age_label, ha='center', va='bottom', fontsize=8
        body_ax.text(3+i, 7, f"{right_val:.2f}", ha='center', va='top', font

# Kulaklar ve veriler
left_ear = Wedge((-2.1, 7), 0.7, 90, 270, color='#e0e0e0', alpha=0.7)
body_ax.add_patch(left_ear)
right_ear = Wedge((2.1, 7), 0.7, -90, 90, color='#e0e0e0', alpha=0.7)
body_ax.add_patch(right_ear)

# İşitme göstergeleri
for i, age_label in enumerate(age_labels):
    if i > 0:  # İlk grup hariç
        left_val = age_group_data.loc[age_label, 'hearing(left)']
        right_val = age_group_data.loc[age_label, 'hearing(right)']

        # İşitme için renk (yüksek değer = kötü işitme)
        left_color = plt.cm.RdYlGn_r(min(1.0, max(0.0, left_val/100)))
        right_color = plt.cm.RdYlGn_r(min(1.0, max(0.0, right_val/100)))

        # İşitme göstergeleri
        left_indicator = Circle((-3-i, 5), 0.3, color=left_color, alpha=0.8)
        body_ax.add_patch(left_indicator)
        body_ax.text(-3-i, 5.5, age_label, ha='center', va='bottom', fontsiz
        body_ax.text(-3-i, 4.5, f"{left_val:.1f} dB", ha='center', va='top',

        right_indicator = Circle((3+i, 5), 0.3, color=right_color, alpha=0.8
        body_ax.add_patch(right_indicator)
        body_ax.text(3+i, 5.5, age_label, ha='center', va='bottom', fontsize
        body_ax.text(3+i, 4.5, f"{right_val:.1f} dB", ha='center', va='top',

# Diş problemleri göstergeleri
# Ağız
mouth = Wedge((0, 5.5), 0.7, 0, 180, color='#e0e0e0', alpha=0.7)
body_ax.add_patch(mouth)

# Yaşa göre diş çürüğü göstergeleri
for i, age_label in enumerate(age_labels):
    if i > 0:
        caries_val = age_group_data.loc[age_label, 'dental caries']
        caries_color = plt.cm.RdYlGn_r(min(1.0, max(0.0, caries_val/5)))

        # Diş çürüğü göstergesi
        caries_indicator = Circle((0, -i-1), 0.3, color=caries_color, alpha=
        body_ax.add_patch(caries_indicator)
        body_ax.text(0, -i-0.5, age_label, ha='center', va='bottom', fontsiz
        body_ax.text(0, -i-1.5, f"Çürük: {caries_val:.1f}", ha='center', va=

# Vücut ana organları göstergeleri
# Kalp
heart = Circle((0, 3), 0.6, color='#d32f2f', alpha=0.7)
body_ax.add_patch(heart)
body_ax.text(0, 3, "♥", ha='center', va='center', fontsize=14, color='white'

# Tansiyon değerleri
for i, age_label in enumerate(age_labels):
    if i > 0:
        systolic_val = data[data['age_group'] == age_label]['systolic'].mean
        diastolic_val = data[data['age_group'] == age_label]['relaxation'].m

        # Tansiyon renk göstergesi
        bp_color = plt.cm.RdYlGn_r(min(1.0, max(0.0, (systolic_val-100)/80)))
```

```
        # Tansiyon göstergesi
        bp_x = 6
        bp_y = -i-1
        bp_indicator = Rectangle((bp_x-0.5, bp_y-0.3), 1, 0.6, color=bp_colo
        body_ax.add_patch(bp_indicator)
        body_ax.text(bp_x, bp_y+0.7, age_label, ha='center', va='bottom', fo
        body_ax.text(bp_x, bp_y, f"{int(systolic_val)}/{int(diastolic_val)}"

# Yaş ve duyusal yetenekler ilişkisi açıklamaları
# Başlık ve görseli tamamlamak için
plt.subplots_adjust(hspace=0.5)
plt.show()
```



In [10]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.colors import LinearSegmentedColormap
import matplotlib.patheffects as path_effects
from matplotlib.patches import Circle, Rectangle, FancyBboxPatch, Ellipse, P
from matplotlib import gridspec
import matplotlib.colors as mcolors

# Load your dataset here (replace with your actual data loading)
# For this example, I'm generating random data
np.random.seed(42)
data = pd.DataFrame({
    'age': np.random.normal(45, 15, 15000),
    'height(cm)': np.random.normal(170, 10, 15000),
    'weight(kg)': np.random.normal(70, 15, 15000),
    'waist(cm)': np.random.normal(85, 15, 15000),
    'systolic': np.random.normal(120, 15, 15000),
    'relaxation': np.random.normal(80, 10, 15000),
    'fasting blood sugar': np.random.normal(95, 20, 15000),
    'Cholesterol': np.random.normal(190, 40, 15000),
    'triglyceride': np.random.normal(150, 80, 15000),
    'HDL': np.random.normal(50, 15, 15000),
    'LDL': np.random.normal(120, 30, 15000),
    'AST': np.random.normal(24, 12, 15000),
    'ALT': np.random.normal(25, 15, 15000),
    'Gtp': np.random.normal(35, 25, 15000)
```

```
})

# Calculate BMI
data['BMI'] = data['weight(kg)'] / ((data['height(cm)']/100) ** 2)

# Create metabolic health score
def metabolic_score(row):
    score = 100  # Starting score
    # Deviations from normal ranges reduce the score
    if row['systolic'] > 130:
        score -= (row['systolic'] - 130) * 0.5
    if row['relaxation'] > 85:
        score -= (row['relaxation'] - 85) * 0.8
    if row['fasting blood sugar'] > 100:
        score -= (row['fasting blood sugar'] - 100) * 0.5
    if row['Cholesterol'] > 200:
        score -= (row['Cholesterol'] - 200) * 0.2
    if row['triglyceride'] > 150:
        score -= (row['triglyceride'] - 150) * 0.1
    if row['LDL'] > 130:
        score -= (row['LDL'] - 130) * 0.3
    if row['HDL'] < 40:
        score -= (40 - row['HDL']) * 0.8
    if row['BMI'] > 25:
        score -= (row['BMI'] - 25) * 2  # Higher penalty for BMI
    if row['ALT'] > 40:
        score -= (row['ALT'] - 40) * 0.5  # Liver enzyme penalty
    return max(0, min(100, score))  # Clamp score between 0 and 100

# Calculate metabolic health score for each person
data['metabolic_score'] = data.apply(metabolic_score, axis=1)

# Determine metabolic syndrome risk category
data['metabolic_risk'] = pd.cut(
    data['metabolic_score'],
    bins=[0, 40, 60, 80, 100],
    labels=['High Risk', 'Moderate Risk', 'Low Risk', 'Optimal'],
    right=True
)

# Determine BMI categories
conditions = [
    (data['BMI'] < 18.5),
    (data['BMI'] >= 18.5) & (data['BMI'] < 25),
    (data['BMI'] >= 25) & (data['BMI'] < 30),
    (data['BMI'] >= 30)
]
values = ['Underweight', 'Normal', 'Overweight', 'Obese']
data['BMI_category'] = np.select(conditions, values)

# Create age groups
age_bins = [0, 30, 40, 50, 60, 100]
age_labels = ['<30', '30-40', '41-50', '51-60', '>60']
data['age_group'] = pd.cut(data['age'], bins=age_bins, labels=age_labels, ri

# Filtreleme: Negatif veya aşırı yüksek yaşları kaldır
data = data[(data['age'] >= 0) & (data['age'] <= 100)]

# --- Visualization: Human Body Metabolic Map ---
plt.style.use('dark_background')
fig = plt.figure(figsize=(18, 16))
fig.patch.set_facecolor('#0d1117')

gs = gridspec.GridSpec(4, 3, height_ratios=[1, 6, 2, 1], width_ratios=[1,1,1
```

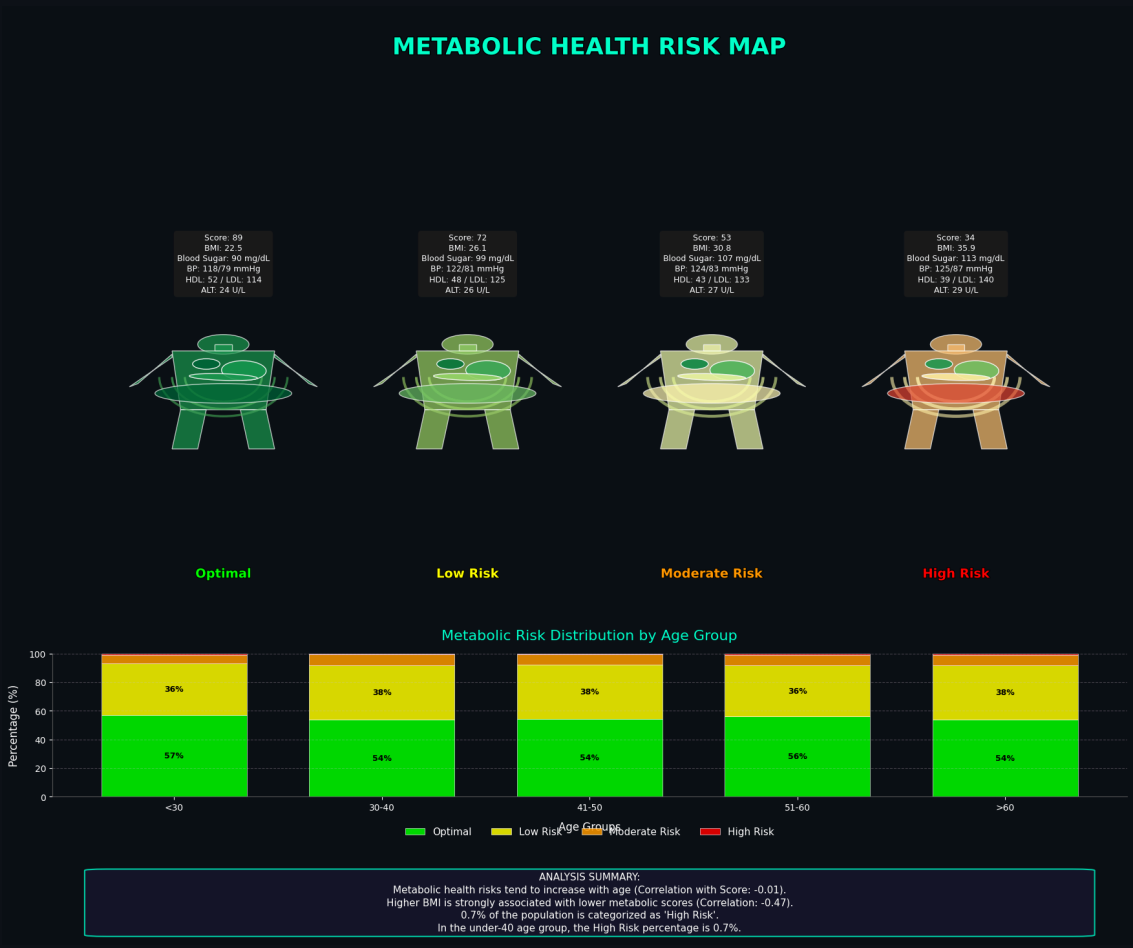
```
# --- Title ---
title_ax = plt.subplot(gs[0, :])
title_ax.axis('off')
title_ax.text(0.5, 0.5, "METABOLIC HEALTH RISK MAP",
              fontsize=26, color='#00ffcc', ha='center', va='center',
              path_effects=[path_effects.withStroke(linewidth=2, foreground=
              fontweight='bold')

# --- Human body and metabolic risk indicator ---
body_ax = plt.subplot(gs[1, :])
body_ax.set_facecolor('#0d1117')
body_ax.set_xlim(-11, 11)
body_ax.set_ylim(-10, 13)
body_ax.axis('off')

# Prepare data grouped by risk category
risk_groups = data.groupby('metabolic_risk', observed=False).agg({
    'systolic': 'mean',
    'relaxation': 'mean',
    'fasting blood sugar': 'mean',
    'Cholesterol': 'mean',
    'triglyceride': 'mean',
    'HDL': 'mean',
    'LDL': 'mean',
    'BMI': 'mean',
    'metabolic_score': 'mean',
    'ALT': 'mean',
    'AST': 'mean',
    'Gtp': 'mean'
}).round(1)

# Risk colors
risk_colors = {
    'Optimal': '#00ff00',
    'Low Risk': '#ffff00',
    'Moderate Risk': '#ff9900',
    'High Risk': '#ff0000'
}

# Reorder risk_groups based on desired display order
risk_order_display = ['Optimal', 'Low Risk', 'Moderate Risk', 'High Risk']
risk_groups = risk_groups.reindex(risk_order_display)

# --- Draw central human body silhouette ---
def draw_human_body_silhouette(ax, highlight_organs=True, scale=1.0, center=
    x, y = center
    # Head
    head = Circle((x, y + 8*scale), 1.5*scale, facecolor='#3a3a3a', edgecolo
    ax.add_patch(head)
    # Neck
    neck = Rectangle((x - 0.5*scale, y + 7*scale), 1*scale, 1*scale, facecol
    ax.add_patch(neck)
    # Torso
    torso = Polygon([(x - 3*scale, y + 7*scale), (x + 3*scale, y + 7*scale),
                     (x + 2.5*scale, y - 2*scale), (x - 2.5*scale, y - 2*sca
                    facecolor='#3a3a3a', edgecolor='white', alpha=0.5)
    ax.add_patch(torso)
    # Arms
    l_arm = Polygon([(x - 3*scale, y + 6.5*scale), (x - 5*scale, y + 2*scale
                     (x - 5.5*scale, y + 1.5*scale), (x - 3*scale, y + 6*sca
                    facecolor='#3a3a3a', edgecolor='white', alpha=0.5)
    ax.add_patch(l_arm)
    r_arm = Polygon([(x + 3*scale, y + 6.5*scale), (x + 5*scale, y + 2*scale
                     (x + 5.5*scale, y + 1.5*scale), (x + 3*scale, y + 6*sca
```

```
                    facecolor='#3a3a3a', edgecolor='white', alpha=0.5)
    ax.add_patch(r_arm)
    # Legs
    l_leg = Polygon([(x - 2.5*scale, y - 2*scale), (x - 1*scale, y - 2*scale
                     (x - 1.5*scale, y - 8*scale), (x - 3*scale, y - 8*scale
                    facecolor='#3a3a3a', edgecolor='white', alpha=0.5)
    ax.add_patch(l_leg)
    r_leg = Polygon([(x + 2.5*scale, y - 2*scale), (x + 1*scale, y - 2*scale
                     (x + 1.5*scale, y - 8*scale), (x + 3*scale, y - 8*scale
                    facecolor='#3a3a3a', edgecolor='white', alpha=0.5)
    ax.add_patch(r_leg)
    if highlight_organs:
        # Brain
        brain = Circle((x, y + 9*scale), 0.9*scale, facecolor='#cc66ff', alp
        ax.add_patch(brain)
        # Heart
        heart = Circle((x - 1*scale, y + 5*scale), 0.8*scale, facecolor='#ff
        ax.add_patch(heart)
        # Liver
        liver = Ellipse((x + 1.2*scale, y + 4*scale), 1.3*scale*2, 1.5*scale
        ax.add_patch(liver)
        # Pancreas
        pancreas = Ellipse((x, y + 3*scale), 2*scale*2, 0.5*scale*2, angle=-
        ax.add_patch(pancreas)
        # Kidneys
        l_kidney = Ellipse((x - 1.8*scale, y + 2.5*scale), 0.7*scale*2, 1*sc
        ax.add_patch(l_kidney)
        r_kidney = Ellipse((x + 1.8*scale, y + 2.5*scale), 0.7*scale*2, 1*sc
        ax.add_patch(r_kidney)
        # Fat tissue (waist area)
        fat_tissue = Ellipse((x, y + 0.5*scale), 4*scale*2, 1.5*scale*2, fac
        ax.add_patch(fat_tissue)
        # Muscle tissue (legs)
        l_muscle = Ellipse((x - 2*scale, y - 4*scale), 1.2*scale*2, 3*scale*
        ax.add_patch(l_muscle)
        r_muscle = Ellipse((x + 2*scale, y - 4*scale), 1.2*scale*2, 3*scale*
        ax.add_patch(r_muscle)

# Draw mini-figures for each risk group
risk_positions = {
    'Optimal': (-7.5, 0),
    'Low Risk': (-2.5, 0),
    'Moderate Risk': (2.5, 0),
    'High Risk': (7.5, 0)
}
mini_fig_scale = 0.35
cmap_health = plt.cm.RdYlGn

for risk, position in risk_positions.items():
    if risk not in risk_groups.index:
        continue
    x_center, y_center = position
    bmi = risk_groups.loc[risk, 'BMI']
    blood_sugar = risk_groups.loc[risk, 'fasting blood sugar']
    cholesterol = risk_groups.loc[risk, 'Cholesterol']
    systolic = risk_groups.loc[risk, 'systolic']
    relaxation = risk_groups.loc[risk, 'relaxation']
    hdl = risk_groups.loc[risk, 'HDL']
    ldl = risk_groups.loc[risk, 'LDL']
    trig = risk_groups.loc[risk, 'triglyceride']
    alt = risk_groups.loc[risk, 'ALT']
    ast = risk_groups.loc[risk, 'AST']
    metabolic_score_val = risk_groups.loc[risk, 'metabolic_score']
```

```
    # Draw Mini Body
    body_color = cmap_health(metabolic_score_val / 100)
    draw_human_body_silhouette(body_ax, highlight_organs=False, scale=mini_f
    for patch in body_ax.patches[-7:]:
        patch.set_facecolor(body_color)
        patch.set_edgecolor('white')
        patch.set_alpha(0.7)

    # Highlight Organs
    heart_health = np.clip(1 - (max(0, systolic - 120)) / 50, 0.1, 1.0)
    heart_color = cmap_health(heart_health)
    mini_heart = Circle((x_center - 1*mini_fig_scale, y_center + 5*mini_fig_
                        facecolor=heart_color, alpha=0.9, edgecolor='white',
    body_ax.add_patch(mini_heart)

    liver_health = np.clip(1 - (max(0, alt - 20)) / 40, 0.1, 1.0)
    liver_color = cmap_health(liver_health)
    mini_liver = Ellipse((x_center + 1.2*mini_fig_scale, y_center + 4*mini_f
                         1.3*mini_fig_scale*2, 1.5*mini_fig_scale*2, angle=1
                         facecolor=liver_color, alpha=0.9, edgecolor='white'
    body_ax.add_patch(mini_liver)

    pancreas_health = np.clip(1 - (max(0, blood_sugar - 85)) / 50, 0.1, 1.0)
    pancreas_color = cmap_health(pancreas_health)
    mini_pancreas = Ellipse((x_center, y_center + 3*mini_fig_scale),
                            2*mini_fig_scale*2, 0.5*mini_fig_scale*2, angle=
                            facecolor=pancreas_color, alpha=0.9, edgecolor='
    body_ax.add_patch(mini_pancreas)

    fat_health = np.clip(1 - (max(0, bmi - 23)) / 15, 0.1, 1.0)
    fat_color = cmap_health(fat_health)
    mini_fat = Ellipse((x_center, y_center + 0.5*mini_fig_scale),
                       4*mini_fig_scale*2, 1.5*mini_fig_scale*2,
                       facecolor=fat_color, alpha=0.7, edgecolor='white', zo
    body_ax.add_patch(mini_fat)

    ldl_penalty = (max(0, ldl - 100)) / 100
    hdl_bonus = (max(0, hdl - 40)) / 40
    trig_penalty = (max(0, trig - 150)) / 200
    vessel_health = np.clip(0.8 + hdl_bonus*0.4 - ldl_penalty*0.6 - trig_per
    vessel_color = cmap_health(vessel_health)

    for i in range(3):
        vessel = Arc((x_center, y_center + 3*mini_fig_scale),
                     (5 - i)*mini_fig_scale*1.5, (8 - i*1.5)*mini_fig_scale*
                     theta1=180, theta2=360,
                     linewidth=2 + (1-vessel_health)*3,
                     color=vessel_color, alpha=0.6 + i*0.1, zorder=8)
        body_ax.add_patch(vessel)

    # Add Text Labels
    body_ax.text(x_center, y_center - 9.5, risk, ha='center', va='center', f
                 fontweight='bold', path_effects=[path_effects.withStroke(li

    info_y_pos = y_center + 5.5
    info_text = (
        f"Score: {metabolic_score_val:.0f}\n"
        f"BMI: {bmi:.1f}\n"
        f"Blood Sugar: {blood_sugar:.0f} mg/dL\n"
        f"BP: {systolic:.0f}/{relaxation:.0f} mmHg\n"
        f"HDL: {hdl:.0f} / LDL: {ldl:.0f}\n"
        f"ALT: {alt:.0f} U/L"
    )
    body_ax.text(x_center, info_y_pos, info_text, ha='center', va='bottom',
                 linespacing=1.3
```

```
                bbox=dict(facecolor='#1f1f1f', alpha=0.7, boxstyle='round,p

# --- Metabolic Risk Distribution Analysis ---
stats_ax = plt.subplot(gs[2, :])
stats_ax.set_facecolor('#0d1117')

# Risk groups by age distribution
risk_by_age = pd.crosstab(data['age_group'], data['metabolic_risk'], normali
risk_by_age = risk_by_age.reindex(columns=risk_order_display, index=age_labe

# Stacked bar chart
bottom_vals = np.zeros(len(age_labels))
bar_width = 0.7

for risk in risk_order_display:
    if risk in risk_by_age.columns:
        values = risk_by_age[risk].values
        stats_ax.bar(age_labels, values, bottom=bottom_vals, color=risk_colo
                     alpha=0.85, label=risk, edgecolor='white', linewidth=0.
        for i, val in enumerate(values):
            if val > 8:
                stats_ax.text(i, bottom_vals[i] + val / 2, f'{val:.0f}%',
                              ha='center', va='center', fontsize=9, color='b
        bottom_vals += values

# Axes and Title adjustments
stats_ax.set_xlabel('Age Groups', fontsize=12, color='white', labelpad=10)
stats_ax.set_ylabel('Percentage (%)', fontsize=12, color='white', labelpad=1
stats_ax.set_title('Metabolic Risk Distribution by Age Group', fontsize=16,
stats_ax.set_ylim(0, 100)
stats_ax.tick_params(axis='x', colors='white', rotation=0)
stats_ax.tick_params(axis='y', colors='white')

# Grid and Spines
stats_ax.grid(True, axis='y', linestyle='--', alpha=0.3, color='#cccccc')
stats_ax.spines['top'].set_visible(False)
stats_ax.spines['right'].set_visible(False)
stats_ax.spines['bottom'].set_color('#555555')
stats_ax.spines['left'].set_color('#555555')

# Legend
stats_ax.legend(loc='upper center', bbox_to_anchor=(0.5, -0.15), ncol=4,
                frameon=False, labelcolor='white', fontsize=11)

# --- Analysis Summary Note ---
analysis_ax = plt.subplot(gs[3, :])
analysis_ax.axis('off')
analysis_ax.set_facecolor('#0d1117')

# Calculate correlations and percentages
valid_data = data[['age', 'metabolic_score', 'BMI']].dropna()
if len(valid_data) > 1:
    age_metabolic_corr = np.corrcoef(valid_data['age'], valid_data['metaboli
    bmi_metabolic_corr = np.corrcoef(valid_data['BMI'], valid_data['metaboli
else:
    age_metabolic_corr = np.nan
    bmi_metabolic_corr = np.nan

high_risk_pct = (data['metabolic_risk'] == 'High Risk').mean() * 100
under_40_data = data[data['age'] < 40]
if not under_40_data.empty:
    young_high_risk = (under_40_data['metabolic_risk'] == 'High Risk').mean(
else:
    young_high_risk = 0
```

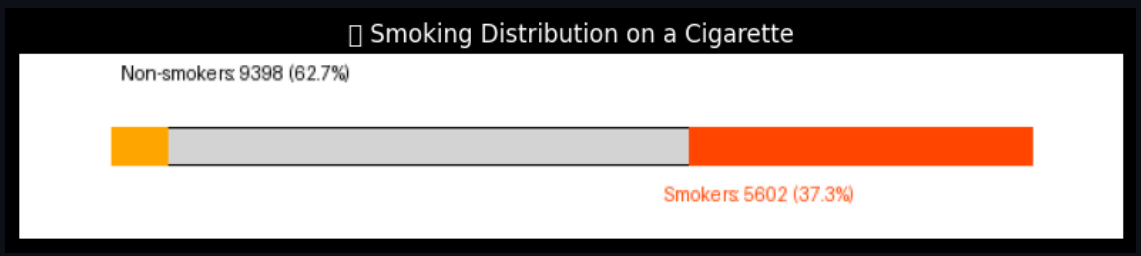
```
# Analysis text
analysis_text = (
    f"ANALYSIS SUMMARY:\n"
    f"Metabolic health risks tend to increase with age (Correlation with Sco
    f"Higher BMI is strongly associated with lower metabolic scores (Correla
    f"{high_risk_pct:.1f}% of the population is categorized as 'High Risk'.\
    f"In the under-40 age group, the High Risk percentage is {young_high_ris
)

# Add a styled text box
analysis_box = FancyBboxPatch((0.05, 0.05), 0.9, 0.9, boxstyle="round,pad=0.
                              facecolor='#1a1a2e', alpha=0.8, ec='#00ffcc', l
                              transform=analysis_ax.transAxes, clip_on=False)
analysis_ax.add_patch(analysis_box)
analysis_ax.text(0.5, 0.5, analysis_text, ha='center', va='center',
                 fontsize=11, color='white', transform=analysis_ax.transAxes
                 linespacing=1.4, wrap=True)

# --- Final Adjustments and Save ---
plt.tight_layout(pad=2.5, h_pad=3.0, rect=[0, 0.03, 1, 0.97])
plt.savefig('metabolic_health_risk_map.png', dpi=300, bbox_inches='tight', f
plt.show()
```



In [11]:

```
from PIL import Image, ImageDraw, ImageFont

non_smokers = 9398
smokers = 5602
total = non_smokers + smokers

non_smoker_pct = non_smokers / total
smoker_pct = smokers / total
```

```
# Görsel boyutu
width = 600
height = 100

# Yeni boş görsel (sigara şekli)
img = Image.new('RGB', (width, height), color='white')
draw = ImageDraw.Draw(img)

# Sigara gövdesi (tam uzunluk)
cigarette_start = (50, 40)
cigarette_end = (550, 60)
draw.rectangle([cigarette_start, cigarette_end], fill='lightgray', outline='

# İçenler oranı: sağdan sola yanık kısmı (kırmızı)
smoke_length = int((cigarette_end[0] - cigarette_start[0]) * smoker_pct)
draw.rectangle(
    [cigarette_end[0] - smoke_length, cigarette_start[1], cigarette_end[0],
    fill='orangered'
)

# Filtre kısmı: solda sabit turuncu alan
filter_width = 30
draw.rectangle(
    [cigarette_start[0], cigarette_start[1], cigarette_start[0] + filter_wid
    fill='orange'
)

# Yüzdeleri yaz
font_size = 16
try:
    font = ImageFont.truetype("arial.ttf", font_size)
except IOError:
    font = ImageFont.load_default()

draw.text((cigarette_start[0] + 5, 5), f"Non-smokers: {non_smokers} ({non_sm
draw.text((cigarette_end[0] - 200, 70), f"Smokers: {smokers} ({smoker_pct:.1


plt.figure(figsize=(10, 2))
plt.imshow(img)
plt.axis('off')
plt.title("🚬 Smoking Distribution on a Cigarette")
plt.show()
```



Exploratory Data Analysis

In [12]:

```
def column_info(df):
```

```
    """
    Generate summary information for each column in the DataFrame.

    Parameters:
    df (DataFrame): Input DataFrame.

    Returns:
    DataFrame: DataFrame containing summary information for each column.
    """
    info = []
    for col in df.columns:
        data_type = df[col].dtype
        count = len(df[col])
        nan_count = df[col].isnull().sum()
        nan_percent = (nan_count / count) * 100 if count > 0 else 0
        unique_count = df[col].nunique()

        if pd.api.types.is_numeric_dtype(df[col]):
            max_value = df[col].max()
            min_value = df[col].min()
            sample_value = df[col].dropna().sample().iloc[0] if count - nan_
        else:
            sample_value = df[col].dropna().sample().iloc[0] if count - nan_
            max_value = 'no value'
            min_value = 'no value'

        info.append({
            'Column_name': col,
            'Data_Type': data_type,
            'Count': count,
            'NaN_Count': nan_count,
            'NaN_Percent': nan_percent,
            'Unique_Count': unique_count,
            'Max_Value': max_value,
            'Min_Value': min_value,
            'Sample_Value': sample_value
        })

    return pd.DataFrame(info)
```

In [13]:

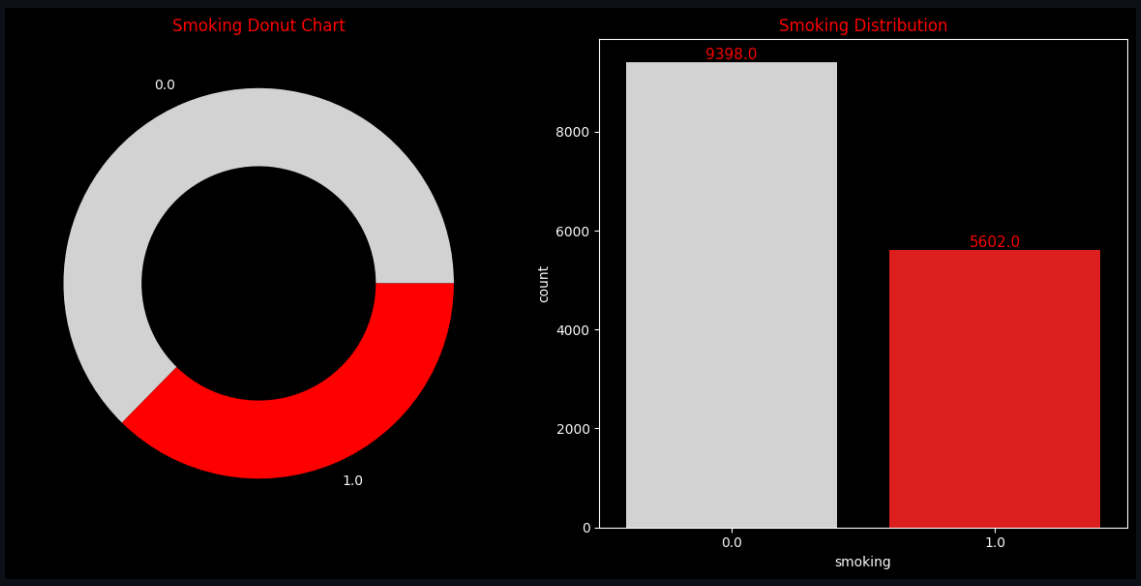
```
column_info(df)
```

Out[13]:

	Column_name	Data_Type	Count	NaN_Count	NaN_Percent	Unique_Count	Max
0	age	float64	5000	0	0.000	5000	
1	systolic	float64	5000	0	0.000	5000	1
2	relaxation	float64	5000	0	0.000	5000	1
3	Cholesterol	float64	5000	0	0.000	5000	3
4	triglyceride	float64	5000	0	0.000	5000	3
5	HDL	float64	5000	0	0.000	5000	
6	LDL	float64	5000	0	0.000	5000	2
7	risk_score	float64	5000	0	0.000	5000	
8	risk_level	category	5000	0	0.000	5	no

In [14]:

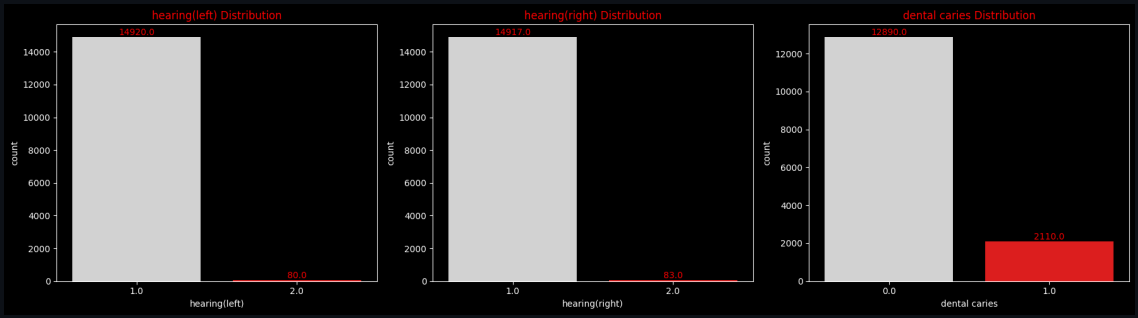
```
df = pd.read_csv("/kaggle/input/binary-smoke-detector/train.csv")

fig, ax = plt.subplots(1, 2, figsize=(12, 6))  # 2M yatay yapı

# --- Sol: Donut Chart ---
smoking_counts = df['smoking'].value_counts()
labels = smoking_counts.index
sizes = smoking_counts.values
colors = ['#d3d3d3', '#ff0000']  # duman grisi ve kırmızı

# Donut chart
wedges, texts = ax[0].pie(sizes, labels=labels, colors=colors, wedgeprops=di
ax[0].set_title('Smoking Donut Chart', color='red')

# --- Sağ: Countplot ---
sns.countplot(x='smoking', data=df, ax=ax[1], palette=['#d3d3d3', '#ff0000']
ax[1].set_title('Smoking Distribution', color='red')

# Sayıları üstüne yaz
for p in ax[1].patches:
    height = p.get_height()
    ax[1].annotate(f'{height}', (p.get_x() + p.get_width() / 2., height),
                   ha='center', va='bottom', fontsize=11, color='red')

plt.tight_layout()
plt.show()
```



In [15]:

```
binary_cols = ['hearing(left)', 'hearing(right)', 'dental caries']

fig, axes = plt.subplots(1, 3, figsize=(18, 5))  # 1 satırda 3 grafik
colors = ['#d3d3d3', '#ff0000']

for i, col in enumerate(binary_cols):
    ax = axes[i]
    sns.countplot(x=col, data=df, palette=colors, ax=ax)
    ax.set_title(f'{col} Distribution', color='red')

    # Sayıları çubuklara yaz
    for p in ax.patches:
        height = p.get_height()
        ax.annotate(f'{height}', (p.get_x() + p.get_width() / 2., height),
```

```
                    ha='center', va='bottom', fontsize=10, color='red')

plt.tight_layout()
plt.show()
```



In [16]:

```
df0 = df.drop(columns="id", axis=1)
df0.columns
```

Out[16]:

```
Index(['age', 'height(cm)', 'weight(kg)', 'waist(cm)', 'eyesight(left)',
       'eyesight(right)', 'hearing(left)', 'hearing(right)', 'systolic',
       'relaxation', 'fasting blood sugar', 'Cholesterol', 'triglyceride',
       'HDL', 'LDL', 'hemoglobin', 'Urine protein', 'serum creatinine', 'AS
T',
       'ALT', 'Gtp', 'dental caries', 'smoking'],
      dtype='object')
```

In [17]:

```
sns.pairplot(data=df, hue="smoking", palette={0.0: "#d3d3d3", 1.0: "#ff0000"
```