```
# 🍽️ FoodieMatch: Análisis de Conversión a Premium
# Growth Hacking para María y Carlos - Ciudad de Panamá

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score, roc_curve
import warnings
warnings.filterwarnings('ignore')

# Configuración de estilo
plt.style.use('seaborn-v0_8-darkgrid')
sns.set_palette("husl")
```

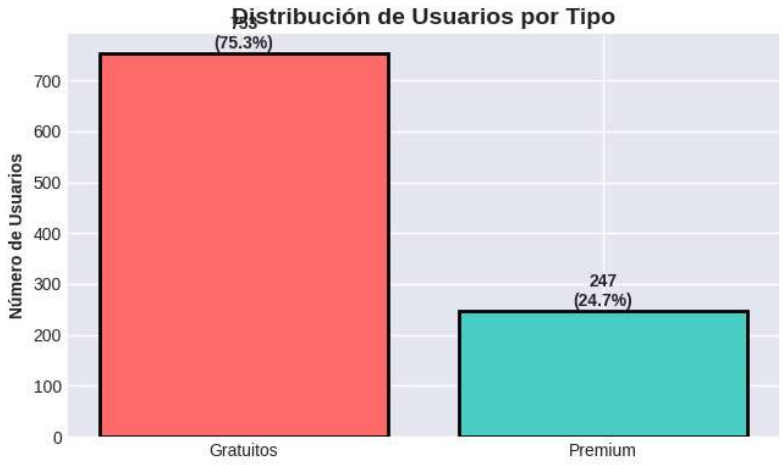
```
# ==========================================
# 1. CARGA Y EXPLORACIÓN INICIAL DE DATOS
# ==========================================

print("="*60)
print("🚀 FOODIEMATCH - ANÁLISIS DE GROWTH HACKING")
print("="*60)
print("\nContexto: App de matching gastronómico en Ciudad de Panamá")
print("Objetivo: Descubrir qué impulsa conversiones a premium")
print("Presupuesto disponible: $10,000")
print("="*60)

# Cargar datos
df = pd.read_csv('/content/foodiematch_dataset.csv')

print("\n📊 RESUMEN INICIAL DE DATOS:")
print("-"*40)
print(f"Total de usuarios analizados: {len(df)}")
print(f"Período de análisis: 3 meses")
print(f"Variables analizadas: {df.shape[1]}")

# Información básica
print("\n📋 Estructura del dataset:")
print(df.info())

print("\n📈 Estadísticas descriptivas:")
print(df.describe().round(2))

# Verificar valores nulos
print("\n🔍 Valores nulos por columna:")
null_counts = df.isnull().sum()
if null_counts.sum() == 0:
    print("✅ No hay valores nulos en el dataset")
else:
    print(null_counts[null_counts > 0])
```

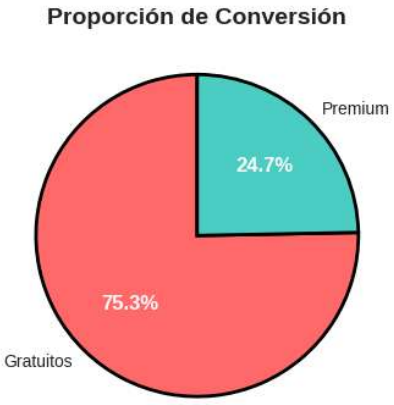
```
----------------------------------------
Total de usuarios analizados: 1000
Período de análisis: 3 meses
Variables analizadas: 10

📋 Estructura del dataset:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 10 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   usuario_id              1000 non-null   int64
 1   sesiones_semana         1000 non-null   int64
 2   matches_realizados      1000 non-null   int64
 3   fotos_subidas           1000 non-null   int64
 4   resenas_escritas        1000 non-null   int64
```

```
       usuario_id  sesiones_semana  matches_realizados  fotos_subidas  \
count     1000.00          1000.00             1000.00        1000.00
mean       500.50             3.92                2.92           2.03
std        288.82             1.99                1.57           1.43
min          1.00             0.00                0.00           0.00
25%        250.75             3.00                2.00           1.00
50%        500.50             4.00                3.00           2.00
75%        750.25             5.00                4.00           3.00
max       1000.00            10.00                9.00           8.00

       resenas_escritas  amigos_invitados  notificaciones_abiertas  \
count           1000.00           1000.00                  1000.00
mean               1.45              0.80                    27.93
std                1.23              0.91                    15.38
min                0.00              0.00                     0.40
25%                1.00              0.00                    15.78
50%                1.00              1.00                    26.35
75%                2.00              1.00                    37.70
max                7.00              5.00                    83.30

       dias_desde_registro     edad  conversion_premium
count              1000.00  1000.00             1000.00
mean                 90.44    27.46                0.25
std                  51.07     4.95                0.43
min                   1.00    18.00                0.00
25%                  46.00    24.00                0.00
50%                  90.00    27.00                0.00
75%                 133.25    31.00                0.00
max                 179.00    44.00                1.00

🔍 Valores nulos por columna:
✅ No hay valores nulos en el dataset
```

```
# ========================================
# 2. ANÁLISIS DE LA VARIABLE OBJETIVO
# ========================================

print("\n" + "="*60)
print("🎯 ANÁLISIS DE CONVERSIÓN A PREMIUM")
print("="*60)

conversion_rate = df['conversion_premium'].mean() * 100
no_premium = len(df[df['conversion_premium'] == 0])
si_premium = len(df[df['conversion_premium'] == 1])

print(f"\n📊 Tasa de conversión actual: {conversion_rate:.2f}%")
print(f"   • Usuarios Gratuitos: {no_premium} ({no_premium/len(df)*100:.1f}%)")
print(f"   • Usuarios Premium: {si_premium} ({si_premium/len(df)*100:.1f}%)")

# Visualización de la distribución
fig, axes = plt.subplots(1, 2, figsize=(12, 4))

# Gráfico de barras
ax1 = axes[0]
colors = ['#FF6B6B', '#4ECDC4']
bars = ax1.bar(['Gratuitos', 'Premium'], [no_premium, si_premium], color=colors, edgecolor='black', linewidth=2)
ax1.set_ylabel('Número de Usuarios', fontweight='bold')
ax1.set_title('Distribución de Usuarios por Tipo', fontweight='bold', fontsize=14)
for bar, val in zip(bars, [no_premium, si_premium]):
    ax1.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 10,
             f'{val}\n({val/len(df)*100:.1f}%)', ha='center', fontweight='bold')

# Gráfico de dona
ax2 = axes[1]
wedges, texts, autotexts = ax2.pie([no_premium, si_premium], labels=['Gratuitos', 'Premium'],
                                   colors=colors, autopct='%1.1f%%', startangle=90,
                                   wedgeprops={'edgecolor': 'black', 'linewidth': 2})
for autotext in autotexts:
    autotext.set_color('white')
    autotext.set_fontweight('bold')
    autotext.set_fontsize(12)
ax2.set_title('Proporción de Conversión', fontweight='bold', fontsize=14)

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

```
============================================================
🎯 ANÁLISIS DE CONVERSIÓN A PREMIUM
============================================================

📊 Tasa de conversión actual: 24.70%
   • Usuarios Gratuitos: 753 (75.3%)
   • Usuarios Premium: 247 (24.7%)
```

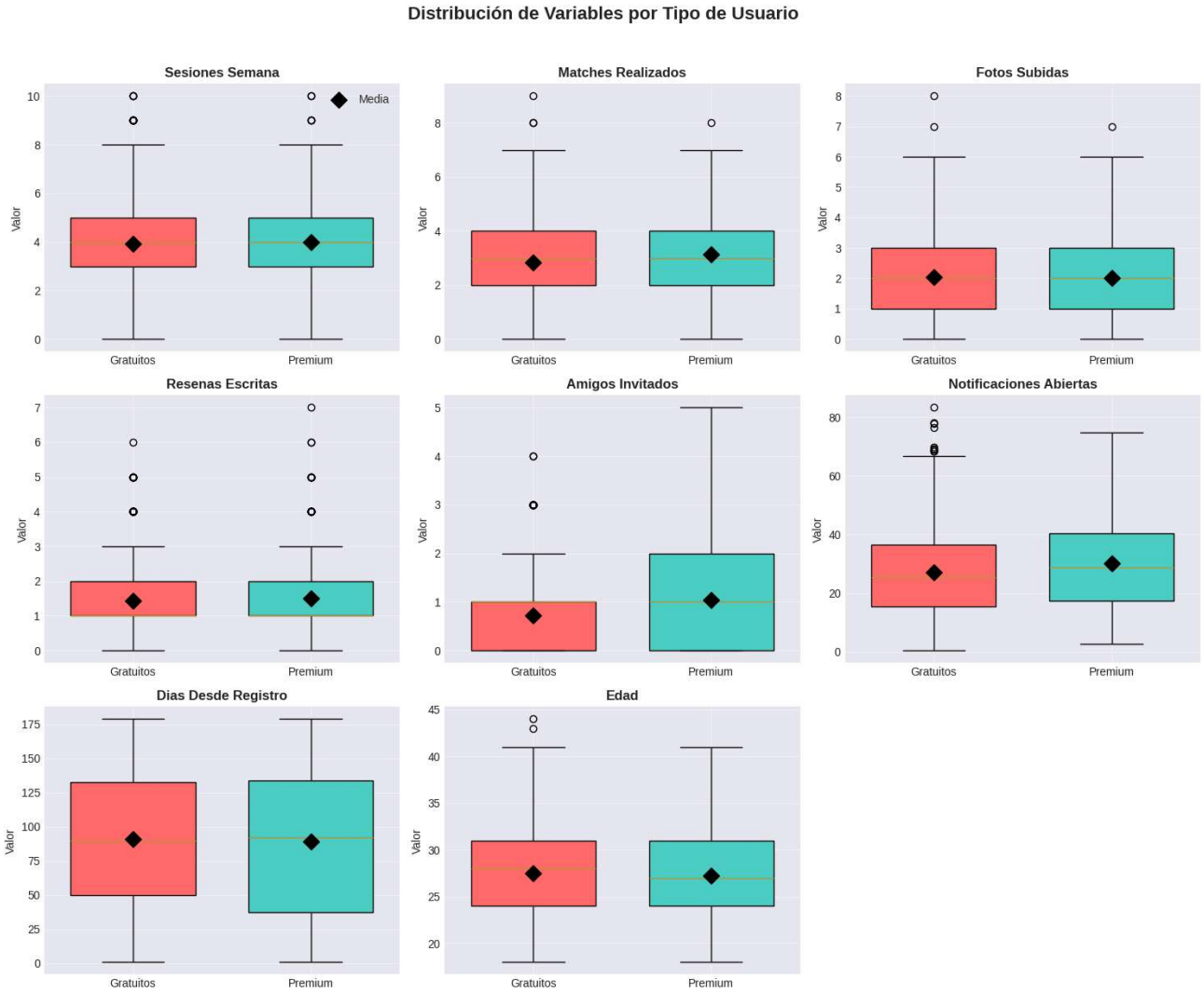





```
# ========================================
# 3. ANÁLISIS EXPLORATORIO POR VARIABLE
# ========================================

print("\n" + "="*60)
print("🔍 ANÁLISIS EXPLORATORIO DE VARIABLES")
print("="*60)

# Comparación de medias entre grupos
print("\n📊 Comparación de medias: Premium vs Gratuitos")
print("-"*50)

variables = ['sesiones_semana', 'matches_realizados', 'fotos_subidas',
             'resenas_escritas', 'amigos_invitados', 'notificaciones_abiertas',
             'dias_desde_registro', 'edad']

comparison_data = []
for var in variables:
    mean_no_premium = df[df['conversion_premium'] == 0][var].mean()
    mean_premium = df[df['conversion_premium'] == 1][var].mean()
    diff_percent = ((mean_premium - mean_no_premium) / mean_no_premium) * 100 if mean_no_premium != 0 else 0

    comparison_data.append({
        'Variable': var,
        'Media Gratuitos': mean_no_premium,
        'Media Premium': mean_premium,
        'Diferencia %': diff_percent
    })

comparison_df = pd.DataFrame(comparison_data)
comparison_df = comparison_df.sort_values('Diferencia %', ascending=False)

for _, row in comparison_df.iterrows():
    emoji = "🚀" if abs(row['Diferencia %']) > 50 else "📈" if abs(row['Diferencia %']) > 20 else "➡️"
    print(f"{emoji} {row['Variable']:<25} | Gratuitos: {row['Media Gratuitos']:>6.2f} | "
          f"Premium: {row['Media Premium']:>6.2f} | Dif: {row['Diferencia %']:>+7.1f}%")

# Visualización de distribuciones
fig, axes = plt.subplots(3, 3, figsize=(15, 12))
axes = axes.ravel()

for idx, var in enumerate(variables):
    ax = axes[idx]

    # Datos por grupo
    data_no_premium = df[df['conversion_premium'] == 0][var]
    data_premium = df[df['conversion_premium'] == 1][var]

    # Boxplot
    bp = ax.boxplot([data_no_premium, data_premium],
                    labels=['Gratuitos', 'Premium'],
                    patch_artist=True,
```

```
                        widths=0.7)

        # Colores
        bp['boxes'][0].set_facecolor('#FF6B6B')
        bp['boxes'][1].set_facecolor('#4ECDC4')

        ax.set_title(f'{var.replace("_", " ").title()}', fontweight='bold')
        ax.set_ylabel('Valor')
        ax.grid(True, alpha=0.3)

        # Agregar medias
        ax.scatter([1, 2], [data_no_premium.mean(), data_premium.mean()],
                   color='black', s=100, zorder=5, marker='D', label='Media')

        if idx == 0:
            ax.legend()

# Eliminar el subplot extra
fig.delaxes(axes[8])

plt.suptitle('Distribución de Variables por Tipo de Usuario', fontsize=16, fontweight='bold', y=1.02)
plt.tight_layout()
plt.show()
```

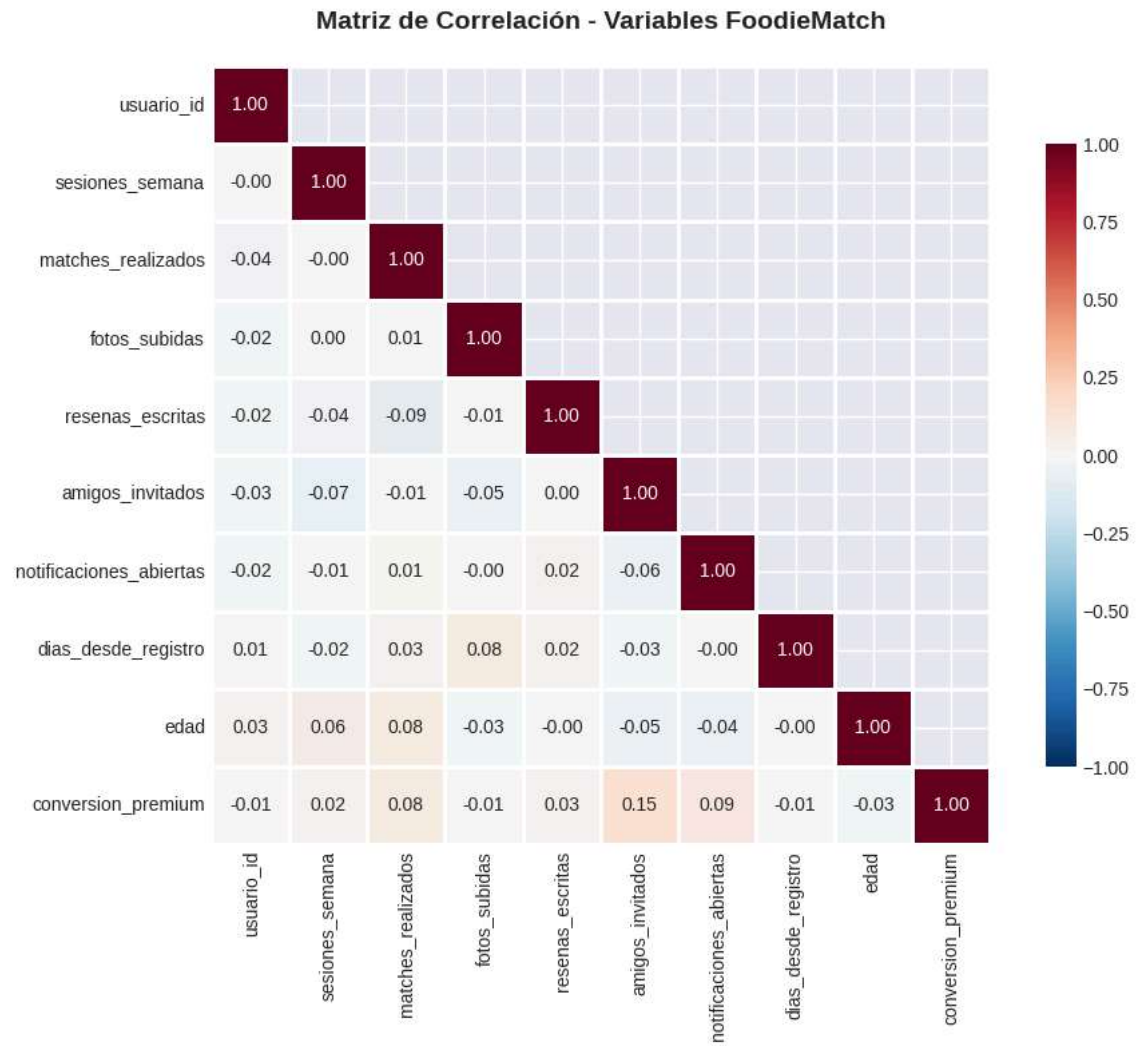
```
============================================================
🔍 ANÁLISIS EXPLORATORIO DE VARIABLES
============================================================

📊 Comparación de medias: Premium vs Gratuitos
--------------------------------------------------
📈 amigos_invitados          | Gratuitos:   0.72 | Premium:   1.04 | Dif:   +44.6%
➡️ notificaciones_abiertas   | Gratuitos:  27.16 | Premium:  30.26 | Dif:   +11.4%
➡️ matches_realizados        | Gratuitos:   2.85 | Premium:   3.13 | Dif:    +9.9%
➡️ resenas_escritas          | Gratuitos:   1.43 | Premium:   1.50 | Dif:    +5.0%
➡️ sesiones_semana           | Gratuitos:   3.90 | Premium:   3.98 | Dif:    +1.9%
➡️ fotos_subidas             | Gratuitos:   2.04 | Premium:   2.02 | Dif:    -0.8%
➡️ edad                      | Gratuitos:  27.54 | Premium:  27.23 | Dif:    -1.1%
➡️ dias_desde_registro       | Gratuitos:  90.83 | Premium:  89.25 | Dif:    -1.7%
```




```
# ========================================
# 4. ANÁLISIS DE CORRELACIÓN
# ========================================

print("\n" + "="*60)
print("📈 ANÁLISIS DE CORRELACIÓN")
print("="*60)

# Matriz de correlación
correlation_matrix = df.corr()

# Correlación con conversion_premium
correlations_with_target = correlation_matrix['conversion_premium'].sort_values(ascending=False)
```

```
print("\n🎯 Correlación con conversión a premium:")
print("-"*40)

for var, corr in correlations_with_target.items():
    if var != 'conversion_premium':
        strength = "Fuerte" if abs(corr) > 0.3 else "Moderada" if abs(corr) > 0.2 else "Débil"
        emoji = "🔥" if abs(corr) > 0.3 else "⚡" if abs(corr) > 0.2 else "💡"
        print(f"{emoji} {var:<25} | r = {corr:>+6.3f} | {strength}")

# Visualización de matriz de correlación
plt.figure(figsize=(10, 8))
mask = np.triu(np.ones_like(correlation_matrix), k=1)
sns.heatmap(correlation_matrix, annot=True, fmt='.2f', cmap='RdBu_r',
            center=0, vmin=-1, vmax=1, mask=mask,
            square=True, linewidths=1, cbar_kws={"shrink": 0.8})
plt.title('Matriz de Correlación - Variables FoodieMatch', fontsize=14, fontweight='bold', pad=20)
plt.tight_layout()
plt.show()
```

```
============================================================
📈 ANÁLISIS DE CORRELACIÓN
============================================================

🎯 Correlación con conversión a premium:
----------------------------------------
💡 amigos_invitados          | r = +0.153 | Débil
💡 notificaciones_abiertas   | r = +0.087 | Débil
💡 matches_realizados        | r = +0.077 | Débil
💡 resenas_escritas          | r = +0.025 | Débil
💡 sesiones_semana           | r = +0.016 | Débil
💡 fotos_subidas             | r = -0.005 | Débil
💡 usuario_id                | r = -0.008 | Débil
💡 dias_desde_registro       | r = -0.013 | Débil
💡 edad                      | r = -0.027 | Débil
```




```
# ========================================
# 5. MODELADO PREDICTIVO
# ========================================

print("\n" + "="*60)
```

```
print("🤖 MODELADO PREDICTIVO")
print("="*60)

# Preparar datos
X = df.drop(['conversion_premium', 'usuario_id'], axis=1)
y = df['conversion_premium']

# División de datos
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)

# Escalado
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

print(f"\n📊 División de datos:")
print(f"   • Entrenamiento: {len(X_train)} usuarios ({len(X_train)/len(df)*100:.1f}%)")
print(f"   • Prueba: {len(X_test)} usuarios ({len(X_test)/len(df)*100:.1f}%)")
```

```
============================================================
🤖 MODELADO PREDICTIVO
============================================================

📊 División de datos:
   • Entrenamiento: 700 usuarios (70.0%)
   • Prueba: 300 usuarios (30.0%)
```

```
# ========================================
# 5.1 REGRESIÓN LOGÍSTICA
# ========================================

print("\n🎯 Modelo 1: Regresión Logística")
print("-"*40)

# Entrenar modelo
log_reg = LogisticRegression(random_state=42, max_iter=1000)
log_reg.fit(X_train_scaled, y_train)

# Predicciones
y_pred_log = log_reg.predict(X_test_scaled)
y_pred_proba_log = log_reg.predict_proba(X_test_scaled)[:, 1]

# Métricas
log_accuracy = log_reg.score(X_test_scaled, y_test)
log_auc = roc_auc_score(y_test, y_pred_proba_log)

print(f"✅ Precisión: {log_accuracy:.3f}")
print(f"✅ AUC-ROC: {log_auc:.3f}")

# Cross-validation
cv_scores_log = cross_val_score(log_reg, X_train_scaled, y_train, cv=5, scoring='roc_auc')
print(f"✅ AUC promedio (CV): {cv_scores_log.mean():.3f} ± {cv_scores_log.std():.3f}")

# Importancia de variables (coeficientes)
feature_importance_log = pd.DataFrame({
    'Variable': X.columns,
    'Coeficiente': log_reg.coef_[0],
    'Abs_Coef': np.abs(log_reg.coef_[0])
}).sort_values('Abs_Coef', ascending=False)

print("\n📊 Top 5 variables más importantes (Reg. Logística):")
for idx, row in feature_importance_log.head().iterrows():
    impact = "positivo" if row['Coeficiente'] > 0 else "negativo"
    print(f"   {idx+1}. {row['Variable']:<25} | Coef: {row['Coeficiente']:>+6.3f} ({impact})")
```

```
🎯 Modelo 1: Regresión Logística
----------------------------------------
✅ Precisión: 0.747
✅ AUC-ROC: 0.528
✅ AUC promedio (CV): 0.644 ± 0.054

📊 Top 5 variables más importantes (Reg. Logística):
   5. amigos_invitados          | Coef: +0.480 (positivo)
   2. matches_realizados        | Coef: +0.225 (positivo)
   6. notificaciones_abiertas   | Coef: +0.185 (positivo)
   7. dias_desde_registro       | Coef: -0.130 (negativo)
   8. edad                      | Coef: -0.101 (negativo)
```

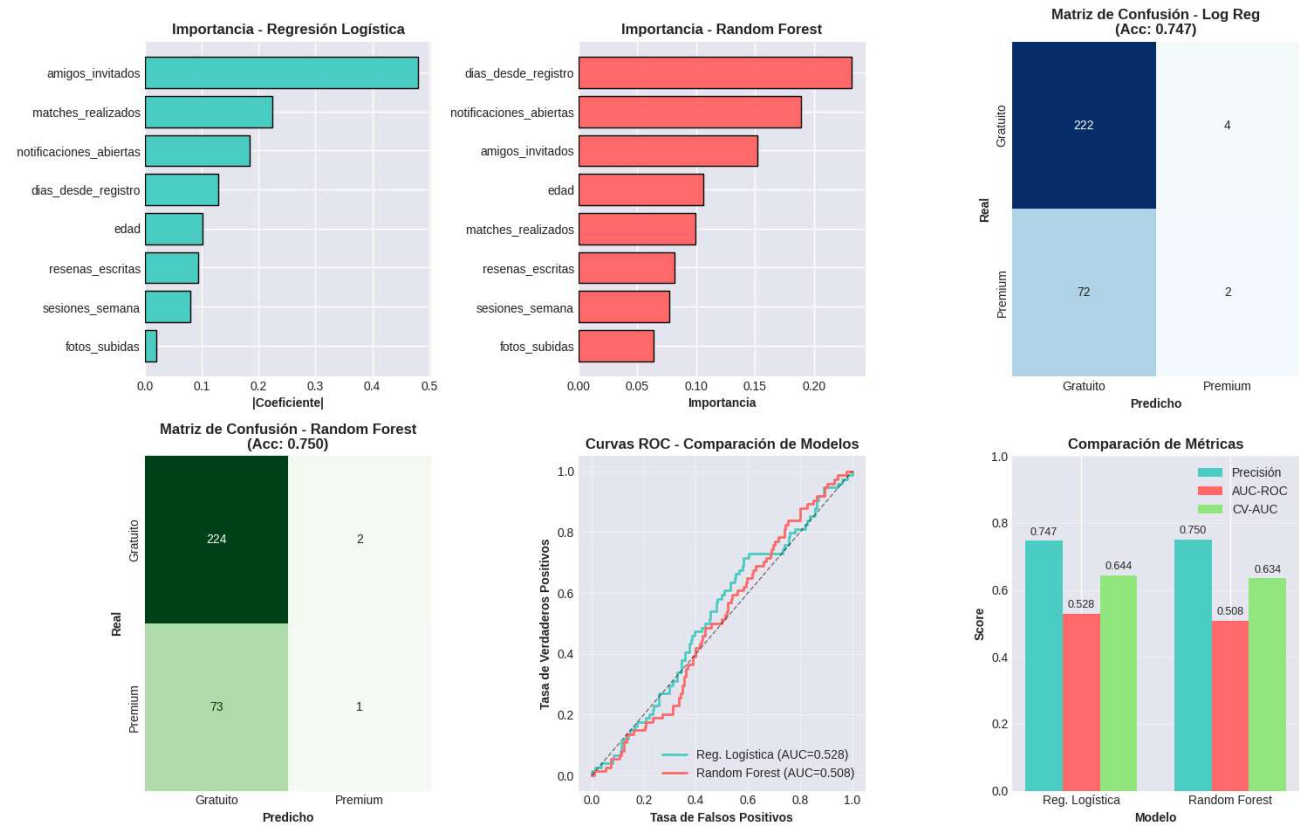
```
# ========================================
# 5.2 RANDOM FOREST
# ========================================
```

```
print("\n🌲 Modelo 2: Random Forest")
print("-"*40)

# Entrenar modelo
rf = RandomForestClassifier(n_estimators=100, random_state=42, max_depth=5)
rf.fit(X_train, y_train)

# Predicciones
y_pred_rf = rf.predict(X_test)
y_pred_proba_rf = rf.predict_proba(X_test)[:, 1]

# Métricas
rf_accuracy = rf.score(X_test, y_test)
rf_auc = roc_auc_score(y_test, y_pred_proba_rf)

print(f"✅ Precisión: {rf_accuracy:.3f}")
print(f"✅ AUC-ROC: {rf_auc:.3f}")

# Cross-validation
cv_scores_rf = cross_val_score(rf, X_train, y_train, cv=5, scoring='roc_auc')
print(f"✅ AUC promedio (CV): {cv_scores_rf.mean():.3f} ± {cv_scores_rf.std():.3f}")

# Importancia de variables
feature_importance_rf = pd.DataFrame({
    'Variable': X.columns,
    'Importancia': rf.feature_importances_
}).sort_values('Importancia', ascending=False)

print("\n📊 Top 5 variables más importantes (Random Forest):")
for idx, row in feature_importance_rf.head().iterrows():
    print(f"   {idx+1}. {row['Variable']:<25} | Importancia: {row['Importancia']:.3f}")
```

```
🌲 Modelo 2: Random Forest
----------------------------------------
✅ Precisión: 0.750
✅ AUC-ROC: 0.508
✅ AUC promedio (CV): 0.634 ± 0.048

📊 Top 5 variables más importantes (Random Forest):
   7. dias_desde_registro       | Importancia: 0.232
   6. notificaciones_abiertas   | Importancia: 0.189
   5. amigos_invitados          | Importancia: 0.152
   8. edad                      | Importancia: 0.106
   2. matches_realizados        | Importancia: 0.099
```

```
# ==========================================
# 6. VISUALIZACIÓN DE RESULTADOS
# ==========================================

fig, axes = plt.subplots(2, 3, figsize=(15, 10))

# 6.1 Importancia de variables - Comparación
ax1 = axes[0, 0]
x_pos = np.arange(len(feature_importance_log))
ax1.barh(x_pos, feature_importance_log['Abs_Coef'].values, color='#4ECDC4', edgecolor='black')
ax1.set_yticks(x_pos)
ax1.set_yticklabels(feature_importance_log['Variable'].values)
ax1.set_xlabel('|Coeficiente|', fontweight='bold')
ax1.set_title('Importancia - Regresión Logística', fontweight='bold')
ax1.invert_yaxis()

ax2 = axes[0, 1]
x_pos = np.arange(len(feature_importance_rf))
ax2.barh(x_pos, feature_importance_rf['Importancia'].values, color='#FF6B6B', edgecolor='black')
ax2.set_yticks(x_pos)
ax2.set_yticklabels(feature_importance_rf['Variable'].values)
ax2.set_xlabel('Importancia', fontweight='bold')
ax2.set_title('Importancia - Random Forest', fontweight='bold')
ax2.invert_yaxis()

# 6.2 Matriz de confusión - Regresión Logística
ax3 = axes[0, 2]
cm_log = confusion_matrix(y_test, y_pred_log)
sns.heatmap(cm_log, annot=True, fmt='d', cmap='Blues', ax=ax3, cbar=False,
            xticklabels=['Gratuito', 'Premium'], yticklabels=['Gratuito', 'Premium'])
ax3.set_title(f'Matriz de Confusión - Log Reg\n(Acc: {log_accuracy:.3f})', fontweight='bold')
ax3.set_ylabel('Real', fontweight='bold')
ax3.set_xlabel('Predicho', fontweight='bold')

# 6.3 Matriz de confusión - Random Forest
```

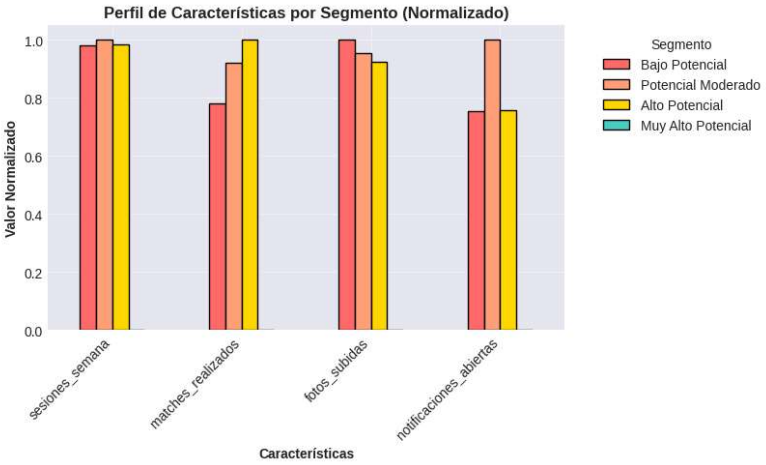
```
ax4 = axes[1, 0]
cm_rf = confusion_matrix(y_test, y_pred_rf)
sns.heatmap(cm_rf, annot=True, fmt='d', cmap='Greens', ax=ax4, cbar=False,
            xticklabels=['Gratuito', 'Premium'], yticklabels=['Gratuito', 'Premium'])
ax4.set_title(f'Matriz de Confusión - Random Forest\n(Acc: {rf_accuracy:.3f})', fontweight='bold')
ax4.set_ylabel('Real', fontweight='bold')
ax4.set_xlabel('Predicho', fontweight='bold')

# 6.4 Curva ROC
ax5 = axes[1, 1]
fpr_log, tpr_log, _ = roc_curve(y_test, y_pred_proba_log)
fpr_rf, tpr_rf, _ = roc_curve(y_test, y_pred_proba_rf)
ax5.plot(fpr_log, tpr_log, label=f'Reg. Logística (AUC={log_auc:.3f})', linewidth=2, color='#4ECDC4')
ax5.plot(fpr_rf, tpr_rf, label=f'Random Forest (AUC={rf_auc:.3f})', linewidth=2, color='#FF6B6B')
ax5.plot([0, 1], [0, 1], 'k--', linewidth=1, alpha=0.5)
ax5.set_xlabel('Tasa de Falsos Positivos', fontweight='bold')
ax5.set_ylabel('Tasa de Verdaderos Positivos', fontweight='bold')
ax5.set_title('Curvas ROC - Comparación de Modelos', fontweight='bold')
ax5.legend(loc='lower right')
ax5.grid(True, alpha=0.3)

# 6.5 Comparación de métricas
ax6 = axes[1, 2]
metrics_comparison = pd.DataFrame({
    'Modelo': ['Reg. Logística', 'Random Forest'],
    'Precisión': [log_accuracy, rf_accuracy],
    'AUC-ROC': [log_auc, rf_auc],
    'CV-AUC': [cv_scores_log.mean(), cv_scores_rf.mean()]
})

x = np.arange(len(metrics_comparison))
width = 0.25

bars1 = ax6.bar(x - width, metrics_comparison['Precisión'], width, label='Precisión', color='#4ECDC4')
bars2 = ax6.bar(x, metrics_comparison['AUC-ROC'], width, label='AUC-ROC', color='#FF6B6B')
bars3 = ax6.bar(x + width, metrics_comparison['CV-AUC'], width, label='CV-AUC', color='#95E77E')

ax6.set_xlabel('Modelo', fontweight='bold')
ax6.set_ylabel('Score', fontweight='bold')
ax6.set_title('Comparación de Métricas', fontweight='bold')
ax6.set_xticks(x)
ax6.set_xticklabels(metrics_comparison['Modelo'])
ax6.legend()
ax6.set_ylim([0, 1])
ax6.grid(True, alpha=0.3, axis='y')

# Agregar valores en las barras
for bars in [bars1, bars2, bars3]:
    for bar in bars:
        height = bar.get_height()
        ax6.text(bar.get_x() + bar.get_width()/2., height + 0.01,
                f'{height:.3f}', ha='center', va='bottom', fontsize=9)

plt.suptitle('Dashboard de Resultados - Modelos Predictivos', fontsize=16, fontweight='bold', y=1.02)
plt.tight_layout()
plt.show()
```




```
# ========================================
# 7. SEGMENTACIÓN DE USUARIOS
# ========================================

print("\n" + "="*60)
print("🎯 SEGMENTACIÓN DE USUARIOS PARA GROWTH HACKING")
print("="*60)

# Calcular probabilidades de conversión para todos los usuarios
df['prob_conversion'] = rf.predict_proba(X)[:, 1]

# Crear segmentos basados en probabilidad
df['segmento'] = pd.cut(df['prob_conversion'],
                        bins=[0, 0.25, 0.5, 0.75, 1.0],
                        labels=['Bajo Potencial', 'Potencial Moderado',
                                'Alto Potencial', 'Muy Alto Potencial'])

print("\n📊 Distribución de usuarios por segmento:")
print("-"*40)
segmento_counts = df['segmento'].value_counts().sort_index()
for segmento, count in segmento_counts.items():
    pct = count/len(df)*100
    print(f"   {segmento:<20} | {count:>4} usuarios ({pct:>5.1f}%)")

# Características por segmento
print("\n🔍 Características promedio por segmento:")
print("-"*50)

key_features = ['sesiones_semana', 'matches_realizados', 'fotos_subidas', 'notificaciones_abiertas']
segmento_profiles = df.groupby('segmento')[key_features].mean()

for segmento in segmento_profiles.index:
    print(f"\n{segmento}:")
    for feature in key_features:
        val = segmento_profiles.loc[segmento, feature]
```

```
            print(f"   • {feature:<25}: {val:.2f}")

    # Visualización de segmentos
    fig, axes = plt.subplots(1, 2, figsize=(14, 5))

    # Distribución de segmentos
    ax1 = axes[0]
    colors_seg = ['#FF6B6B', '#FFA07A', '#FFD700', '#4ECDC4']
    wedges, texts, autotexts = ax1.pie(segmento_counts.values, labels=segmento_counts.index,
                                       colors=colors_seg, autopct='%1.1f%%', startangle=90,
                                       wedgeprops={'edgecolor': 'black', 'linewidth': 2})
    for autotext in autotexts:
        autotext.set_color('white')
        autotext.set_fontweight('bold')
    ax1.set_title('Segmentación de Usuarios por Potencial de Conversión', fontweight='bold', fontsize=12)

    # Características por segmento (radar chart simplificado)
    ax2 = axes[1]
    segmento_profiles_norm = segmento_profiles / segmento_profiles.max()
    segmento_profiles_norm.T.plot(kind='bar', ax=ax2, color=colors_seg, edgecolor='black')
    ax2.set_title('Perfil de Características por Segmento (Normalizado)', fontweight='bold', fontsize=12)
    ax2.set_xlabel('Características', fontweight='bold')
    ax2.set_ylabel('Valor Normalizado', fontweight='bold')
    ax2.legend(title='Segmento', bbox_to_anchor=(1.05, 1), loc='upper left')
    ax2.set_xticklabels(ax2.get_xticklabels(), rotation=45, ha='right')
    ax2.grid(True, alpha=0.3)

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

```
============================================================
🎯 SEGMENTACIÓN DE USUARIOS PARA GROWTH HACKING
============================================================

📊 Distribución de usuarios por segmento:
----------------------------------------
   Bajo Potencial       |  644 usuarios ( 64.4%)
   Potencial Moderado   |  334 usuarios ( 33.4%)
   Alto Potencial       |   22 usuarios (  2.2%)
   Muy Alto Potencial   |    0 usuarios (  0.0%)

🔍 Características promedio por segmento:
--------------------------------------------------

Bajo Potencial:
   • sesiones_semana          : 3.90
   • matches_realizados       : 2.74
   • fotos_subidas            : 2.07
   • notificaciones_abiertas  : 25.19

Potencial Moderado:
   • sesiones_semana          : 3.97
   • matches_realizados       : 3.22
   • fotos_subidas            : 1.98
   • notificaciones_abiertas  : 33.38

Alto Potencial:
   • sesiones_semana          : 3.91
   • matches_realizados       : 3.50
   • fotos_subidas            : 1.91
   • notificaciones_abiertas  : 25.34

Muy Alto Potencial:
   • sesiones_semana          : nan
   • matches_realizados       : nan
   • fotos_subidas            : nan
   • notificaciones_abiertas  : nan
```




```
# ============================================
# 8. RECOMENDACIONES DE GROWTH HACKING
# ============================================

print("\n" + "="*60)
print("💡 RECOMENDACIONES DE GROWTH HACKING")
print("="*60)

print("\n🎯 INSIGHTS CLAVE DESCUBIERTOS:")
print("-"*40)

# Identificar las top 3 variables más importantes
top_features = feature_importance_rf.head(3)

print("\n1️⃣ FACTORES CRÍTICOS DE CONVERSIÓN:")
for idx, row in top_features.iterrows():
    feature = row['Variable']
    importance = row['Importancia']
    mean_diff = comparison_df[comparison_df['Variable'] == feature]['Diferencia %'].values[0]

    print(f"\n   🔥 {feature.replace('_', ' ').title()}:")
    print(f"      • Importancia: {importance:.1%}")
```

```
        print(f"      • Usuarios Premium tienen {abs(mean_diff):.0f}% {'más' if mean_diff > 0 else 'menos'} que Gratuitos")

print("\n2️⃣ ESTRATEGIAS RECOMENDADAS CON PRESUPUESTO $10,000:")
print("-"*40)

# Estrategias basadas en los insights
strategies = []

# Analizar variable más importante
top_var = top_features.iloc[0]['Variable']
if top_var == 'matches_realizados':
    strategies.append({
        'estrategia': 'Campaña de Matching Acelerado',
        'presupuesto': 3000,
        'descripcion': 'Eventos presenciales de speed-dining y mejora del algoritmo de matching',
        'roi_esperado': '250%'
    })
elif top_var == 'fotos_subidas':
    strategies.append({
        'estrategia': 'Concurso de Fotografía Gastronómica',
        'presupuesto': 2500,
        'descripcion': 'Premios para las mejores fotos y partnerships con restaurantes',
        'roi_esperado': '200%'
    })
elif top_var == 'sesiones_semana':
    strategies.append({
        'estrategia': 'Programa de Engagement Diario',
        'presupuesto': 3500,
        'descripcion': 'Notificaciones personalizadas y daily challenges gastronómicos',
        'roi_esperado': '180%'
    })

# Estrategias adicionales basadas en segmentación
strategies.extend([
```