Exploring Mental Health Data

- **Dataset**: https://www.kaggle.com/competitions/playground-series-s4e11/data

Importar dependencias

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
C:\Users\34644\anaconda3\lib\site-packages\pandas\core\computation\expressions.py:21: UserWarning: Pandas requires version '2.8.4' or newer of 'nu
mexpr' (version '2.8.3' currently installed).
  from pandas.core.computation.check import NUMEXPR_INSTALLED
C:\Users\34644\anaconda3\lib\site-packages\pandas\core\arrays\masked.py:60: UserWarning: Pandas requires version '1.3.6' or newer of 'bottleneck'
(version '1.3.5' currently installed).
  from pandas.core import (
```

In [3]:
```
base_path = "../../Datasets/depression/"
```

Explorando los datos

In [4]:
```
df = pd.read_csv(f"{base_path}train.csv")
```

In [5]:
```
df.shape
```

Out[5]:
```
(140700, 20)
```

In [6]:
```
df
```

Out[6]:

	id	Name	Gender	Age	City	Working Professional or Student	Profession	Academic Pressure	Work Pressure	CGPA	Study Satisfaction	Job Satisfaction	Sleep Duration	Dietary Habits	Degree	Have you ever had suicidal thoughts ?	Work/Stu Ho
0	0	Aaradhya	Female	49.0	Ludhiana	Working Professional	Chef	NaN	5.0	NaN	NaN	2.0	More than 8 hours	Healthy	BHM	No	
1	1	Vivan	Male	26.0	Varanasi	Working Professional	Teacher	NaN	4.0	NaN	NaN	3.0	Less than 5 hours	Unhealthy	LLB	Yes	
2	2	Yuvraj	Male	33.0	Visakhapatnam	Student	NaN	5.0	NaN	8.97	2.0	NaN	5-6 hours	Healthy	B.Pharm	Yes	
3	3	Yuvraj	Male	22.0	Mumbai	Working Professional	Teacher	NaN	5.0	NaN	NaN	1.0	Less than 5 hours	Moderate	BBA	Yes	1
4	4	Rhea	Female	30.0	Kanpur	Working Professional	Business Analyst	NaN	1.0	NaN	NaN	1.0	5-6 hours	Unhealthy	BBA	Yes	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
140695	140695	Vidya	Female	18.0	Ahmedabad	Working Professional	NaN	NaN	5.0	NaN	NaN	4.0	5-6 hours	Unhealthy	Class 12	No	
140696	140696	Lata	Female	41.0	Hyderabad	Working Professional	Content Writer	NaN	5.0	NaN	NaN	4.0	7-8 hours	Moderate	B.Tech	Yes	
140697	140697	Aanchal	Female	24.0	Kolkata	Working Professional	Marketing Manager	NaN	3.0	NaN	NaN	1.0	More than 8 hours	Moderate	B.Com	No	
140698	140698	Prachi	Female	49.0	Srinagar	Working Professional	Plumber	NaN	5.0	NaN	NaN	2.0	5-6 hours	Moderate	ME	Yes	1
140699	140699	Sai	Male	27.0	Patna	Student	NaN	4.0	NaN	9.24	1.0	NaN	Less than 5 hours	Healthy	BCA	Yes	

140700 rows × 20 columns

Identificando las columnas existentes

In [7]:
```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 140700 entries, 0 to 140699
Data columns (total 20 columns):
 #   Column                                 Non-Null Count   Dtype  
---  ------                                 --------------   -----  
 0   id                                     140700 non-null  int64  
 1   Name                                   140700 non-null  object 
 2   Gender                                 140700 non-null  object 
 3   Age                                    140700 non-null  float64
 4   City                                   140700 non-null  object 
 5   Working Professional or Student        140700 non-null  object 
 6   Profession                             104070 non-null  object 
 7   Academic Pressure                      27897 non-null   float64
 8   Work Pressure                          112782 non-null  float64
 9   CGPA                                   27898 non-null   float64
 10  Study Satisfaction                     27897 non-null   float64
 11  Job Satisfaction                       112790 non-null  float64
 12  Sleep Duration                         140700 non-null  object 
 13  Dietary Habits                         140696 non-null  object 
 14  Degree                                 140698 non-null  object 
 15  Have you ever had suicidal thoughts ?  140700 non-null  object 
 16  Work/Study Hours                       140700 non-null  float64
 17  Financial Stress                       140696 non-null  float64
 18  Family History of Mental Illness       140700 non-null  object 
 19  Depression                             140700 non-null  int64  
dtypes: float64(8), int64(2), object(10)
memory usage: 21.5+ MB
None
```

Explorando columnas vacías

In [7]:
```
print(df.isnull().sum())
```

```
id                                            0
Name                                          0
Gender                                        0
Age                                           0
City                                          0
Working Professional or Student               0
Profession                                36630
Academic Pressure                        112803
Work Pressure                             27918
CGPA                                     112802
Study Satisfaction                       112803
Job Satisfaction                          27910
Sleep Duration                                0
Dietary Habits                                4
Degree                                        2
Have you ever had suicidal thoughts ?         0
Work/Study Hours                              0
Financial Stress                              4
Family History of Mental Illness              0
Depression                                    0
dtype: int64
```

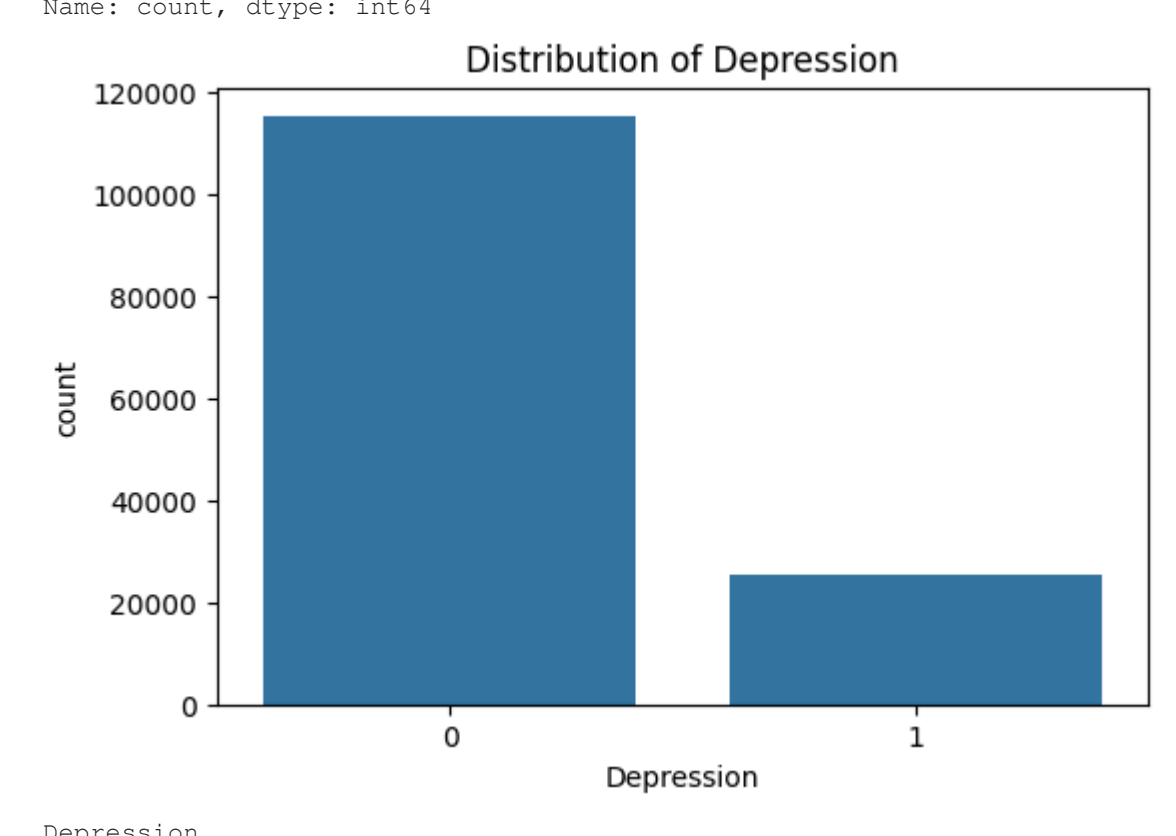
In [8]:
```
print(df.dtypes)
```

```
id                                         int64
Name                                      object
Gender                                    object
Age                                      float64
City                                      object
Working Professional or Student           object
Profession                                object
Academic Pressure                        float64
Work Pressure                            float64
CGPA                                     float64
Study Satisfaction                       float64
Job Satisfaction                         float64
Sleep Duration                            object
Dietary Habits                            object
Degree                                    object
Have you ever had suicidal thoughts ?     object
Work/Study Hours                         float64
Financial Stress                         float64
Family History of Mental Illness          object
Depression                                 int64
dtype: object
```

Explorando columna objetivo

In [8]:
```
print(df['Depression'].value_counts())
plt.figure(figsize=(6, 4))
sns.countplot(x='Depression', data=df)
plt.title('Distribution of Depression')
plt.show()
print((df['Depression'].value_counts(normalize=True) * 100).round())
```

```
Depression
0    115133
1     25567
Name: count, dtype: int64
```




```
Depression
0    82.0
1    18.0
Name: proportion, dtype: float64
```

Explorando columnas categóricas

In [10]:
```
categorical_cols = df.select_dtypes(include=['object']).columns
for col in categorical_cols:
    print(f'Unique values in {col}: {df[col].nunique()}')
```

```
Unique values in Name: 422
Unique values in Gender: 2
Unique values in City: 98
Unique values in Working Professional or Student: 2
Unique values in Profession: 64
Unique values in Sleep Duration: 36
Unique values in Dietary Habits: 23
Unique values in Degree: 115
Unique values in Have you ever had suicidal thoughts ?: 2
Unique values in Family History of Mental Illness: 2
```

Preprocesando los datos

Rellenamos los valores nulos de las columnas **numéricas** con la **media** y las **categóricas** con la **moda** con la intención de que distorsionen lo menos posible los datos

In [9]:
```
def preprocess_data(df: pd.DataFrame) -> pd.DataFrame:
   df = df.copy()

   num_cols = df.select_dtypes('number').columns
   df[num_cols] = df[num_cols].fillna(df[num_cols].mean())

   cat_cols = df.select_dtypes('object').columns
   for col in cat_cols:
       df[col] = df[col].fillna(df[col].mode()[0])

   return df
```

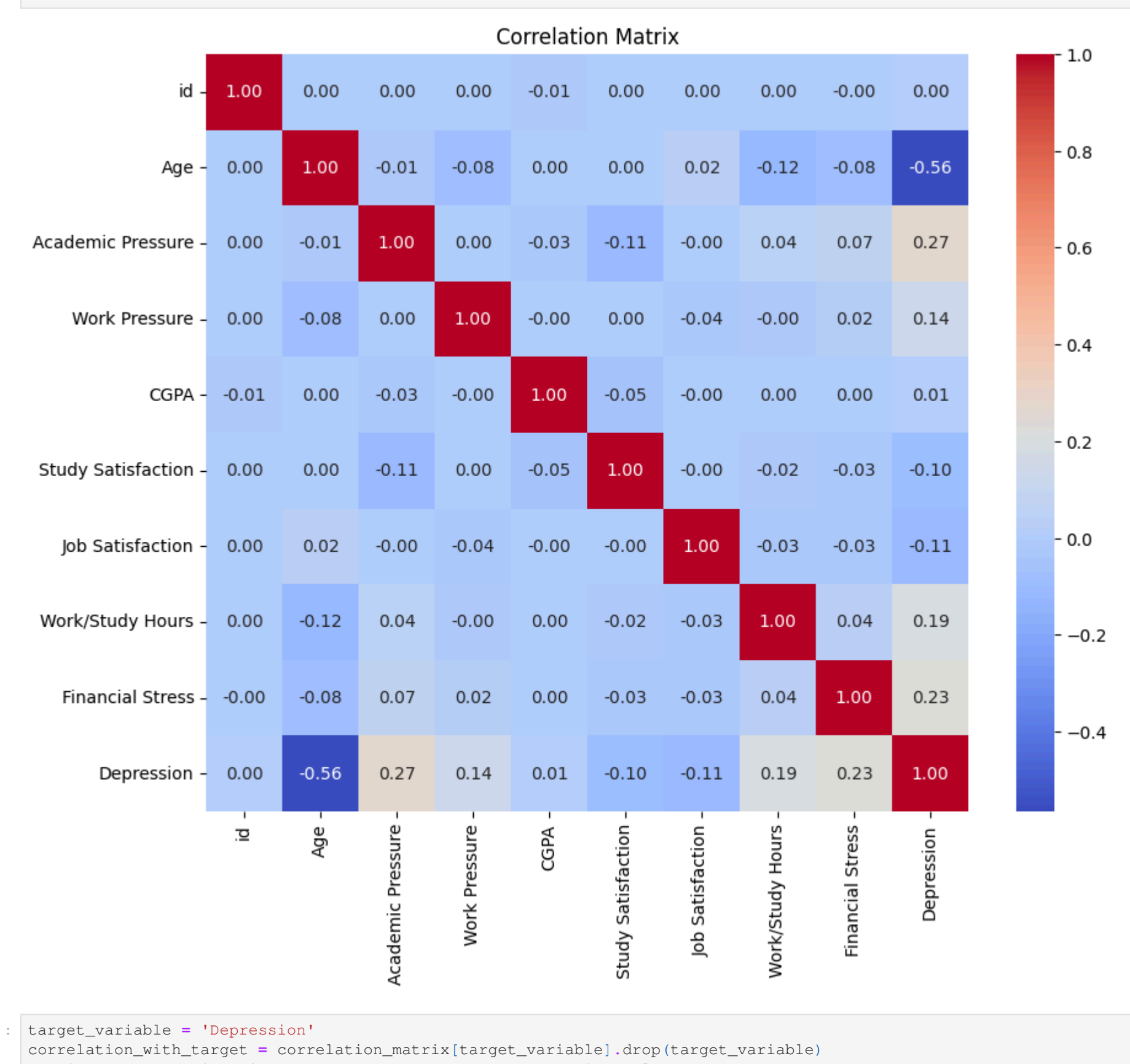
In [10]:
```
df = preprocess_data(df)

numerical_cols = df.select_dtypes(include=['number']).columns
categorical_cols = df.select_dtypes(include=['object']).columns

print(df.describe())
```

```
                  id            Age  Academic Pressure  Work Pressure  \
count  140700.000000  140700.000000      140700.000000  140700.000000   
mean    70349.500000      40.388621           3.142273       2.998998   
std     40616.735775      12.384099           0.614679       1.258598   
min         0.000000      18.000000           1.000000       1.000000   
25%     35174.750000      29.000000           3.142273       2.000000   
50%     70349.500000      42.000000           3.142273       2.998998   
75%    105524.250000      51.000000           3.142273       4.000000   
max    140699.000000      60.000000           5.000000       5.000000   

                CGPA  Study Satisfaction  Job Satisfaction  Work/Study Hours  \
count  140700.000000       140700.000000     140700.000000     140700.000000   
mean        7.658636            2.944940          2.974404          6.252679   
std         0.652098            0.605658          1.267871          3.853615   
min         5.030000            1.000000          1.000000          0.000000   
25%         7.658636            2.944940          2.000000          3.000000   
50%         7.658636            2.944940          2.974404          6.000000   
75%         7.658636            2.944940          4.000000         10.000000   
max        10.000000            5.000000          5.000000         12.000000   

       Financial Stress     Depression  
count     140700.000000  140700.000000  
mean           2.988983       0.181713  
std            1.413613       0.385609  
min            1.000000       0.000000  
25%            2.000000       0.000000  
50%            3.000000       0.000000  
75%            4.000000       0.000000  
max            5.000000       1.000000  
```

Tras hacer un .describe para ver la información estadística básica de las columnas observamos que el parámetro que tiene una media mayor es **"Academic Pressure"**

Hacemos un correlograma para comprobar cuanto peso tendrá esta variable en el modelo:

In [11]:
```
plt.figure(figsize=(10, 8))
correlation_matrix = df[numerical_cols].corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix')
plt.show()
```



In [22]:
```
target_variable = 'Depression'
correlation_with_target = correlation_matrix[target_variable].drop(target_variable)
ranking = correlation_with_target.abs().sort_values(ascending=False)
ranking = ranking[:3].round(2)
print("Ranking de variables más relevantes:")
print(ranking)
```

```
Ranking de variables más relevantes:
Age                  0.56
Academic Pressure    0.27
Financial Stress     0.23
Name: Depression, dtype: float64
```

Comprobamos que "Academic Pressure" es la segunda variable más importante después de "Financial Stress". Sin embargo la variable más representativa es **"Age"**, que nos hace llegar a la conclusión de que a cuanto mayor sea la edad del encuestado menor será la probabilidad de que tenga depresión

In [12]:
```
#Generamos las dummies de las variables categóricas para poder entrenar los modelos

categorical_cols = ['Gender', 'City', 'Working Professional or Student', 'Profession', 'Sleep Duration', 'Dietary Habits', 'Degree', 'Have you ever

df_encoded = pd.get_dummies(df, columns=categorical_cols, drop_first=True)
```

Train Test Split

Dividimos el dataframe en train y test y escalamos las variables numéricas

In [13]:
```
from sklearn.model_selection import train_test_split

X = df_encoded.drop('Depression', axis=1)
y = df_encoded['Depression']

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

print("X_train shape:", X_train.shape)
print("y_train shape:", y_train.shape)
print("X_test shape:", X_test.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (112560, 345)
y_train shape: (112560,)
X_test shape: (28140, 345)
y_test shape: (28140,)
```

In [14]:
```
def add_squared_features_and_scale(X_train, X_test, numerical_features):
   from sklearn.preprocessing import StandardScaler

   for feature in numerical_features:
       X_train[feature + '_squared'] = X_train[feature] ** 2
       X_test[feature + '_squared'] = X_test[feature] ** 2

   scaler = StandardScaler()
   X_train[numerical_features] = scaler.fit_transform(X_train[numerical_features])
   X_test[numerical_features] = scaler.transform(X_test[numerical_features])

   return X_train, X_test, scaler
```

In [15]:
```
numerical_features = ['Age', 'CGPA', 'Study Satisfaction', 'Job Satisfaction', 'Work/Study Hours', 'Financial Stress']

X_train, X_test, scaler = add_squared_features_and_scale(X_train, X_test, numerical_features)
```

In [16]:
```
X_train = X_train.drop(['Name', 'id'], axis=1)
X_test = X_test.drop(['Name', 'id'], axis=1)

print(X_train.shape)
print(X_test.shape)
```

```
(112560, 349)
(28140, 349)
```

In [17]:
```
X_train.head()
```

Out[17]:

	Age	Academic Pressure	Work Pressure	CGPA	Study Satisfaction	Job Satisfaction	Work/Study Hours	Financial Stress	Gender_Male	City_Aaradhya	...	Degree_Vrinda	Degree_Working Professional	Have you ever had suicidal thoughts ?_Yes	Fam History Men Illness_Y
3429	-1.241462	5.000000	2.998998	-3.166512	3.388457	0.001574	0.457470	1.423178	False	False	...	False	False	True	Fal
57741	-1.644762	3.000000	2.998998	0.935933	1.738588	0.001574	-1.617987	-1.406348	False	False	...	False	False	False	Tr
83234	-1.322122	3.000000	2.998998	-2.538899	-1.561149	0.001574	-0.839690	-1.406348	True	False	...	False	False	False	Tr
136573	-0.192882	3.142273	1.000000	0.000080	-0.002122	0.021752	0.976335	-0.698967	False	False	...	False	False	False	Tr
93261	-1.322122	4.000000	2.998998	0.583858	1.738588	0.001574	0.976335	0.008415	False	False	...	False	False	True	Tr

5 rows × 349 columns

Evaluando modelos

In [18]:
```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, recall_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
```

Creamos una matriz de confusión para poder comprobar el desempeño de los modelos y compararlos. En problemas de calsificación nos será de gran ayuda, ya que podremos saber exactamente cuantos falsos negativos tenemos en cada modelo (recall)

In [19]:
```
def plot_confusion_matrix(y_true, y_pred, figsize=(8, 6), title='Confusion Matrix'):
   cm = confusion_matrix(y_true, y_pred)
   plt.figure(figsize=figsize)
   sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
               xticklabels=['Predicted 0', 'Predicted 1'],
               yticklabels=['Actual 0', 'Actual 1'])
   plt.title(title)
   plt.xlabel('Predicted')
   plt.ylabel('Actual')
   plt.show()
```

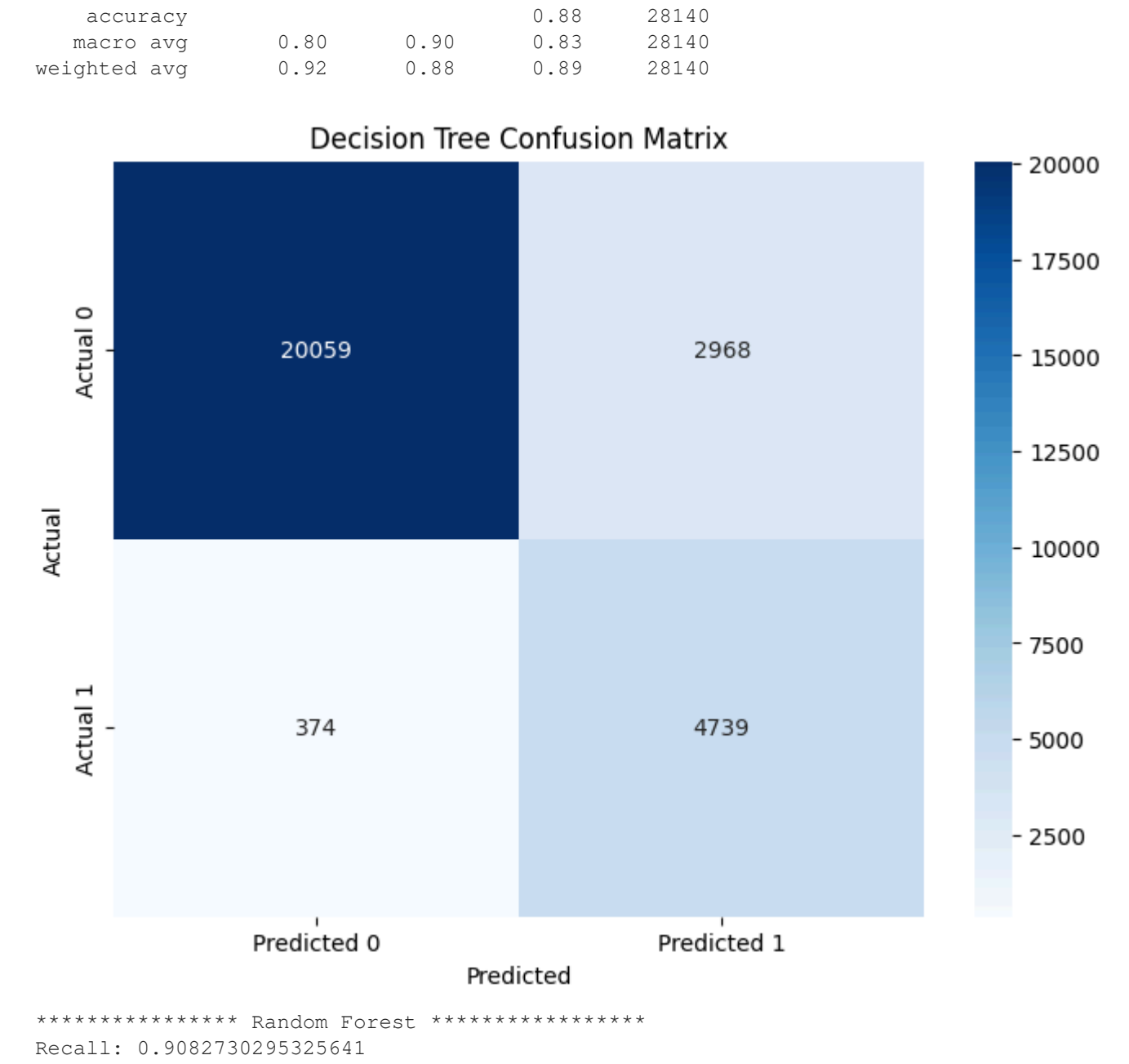
En las dos siguientes celdas asignaremos un *classification_report*

In [20]:
```
def evaluate_model(model, X_train, y_train, X_test, y_test, name):
   fitted_model = model.fit(X_train, y_train)
   predictions = model.predict(X_test)
   recall = recall_score(y_test, predictions)

   print(f"\n**************** {name} *****************")
   print(f"Recall: {recall}")
   print("--------------------------------------------")
   print("Classification Report \n")
   print(classification_report(y_test, predictions))

   plot_confusion_matrix(y_test, predictions, title=f"{name} Confusion Matrix")

   return (fitted_model, predictions, recall)
```

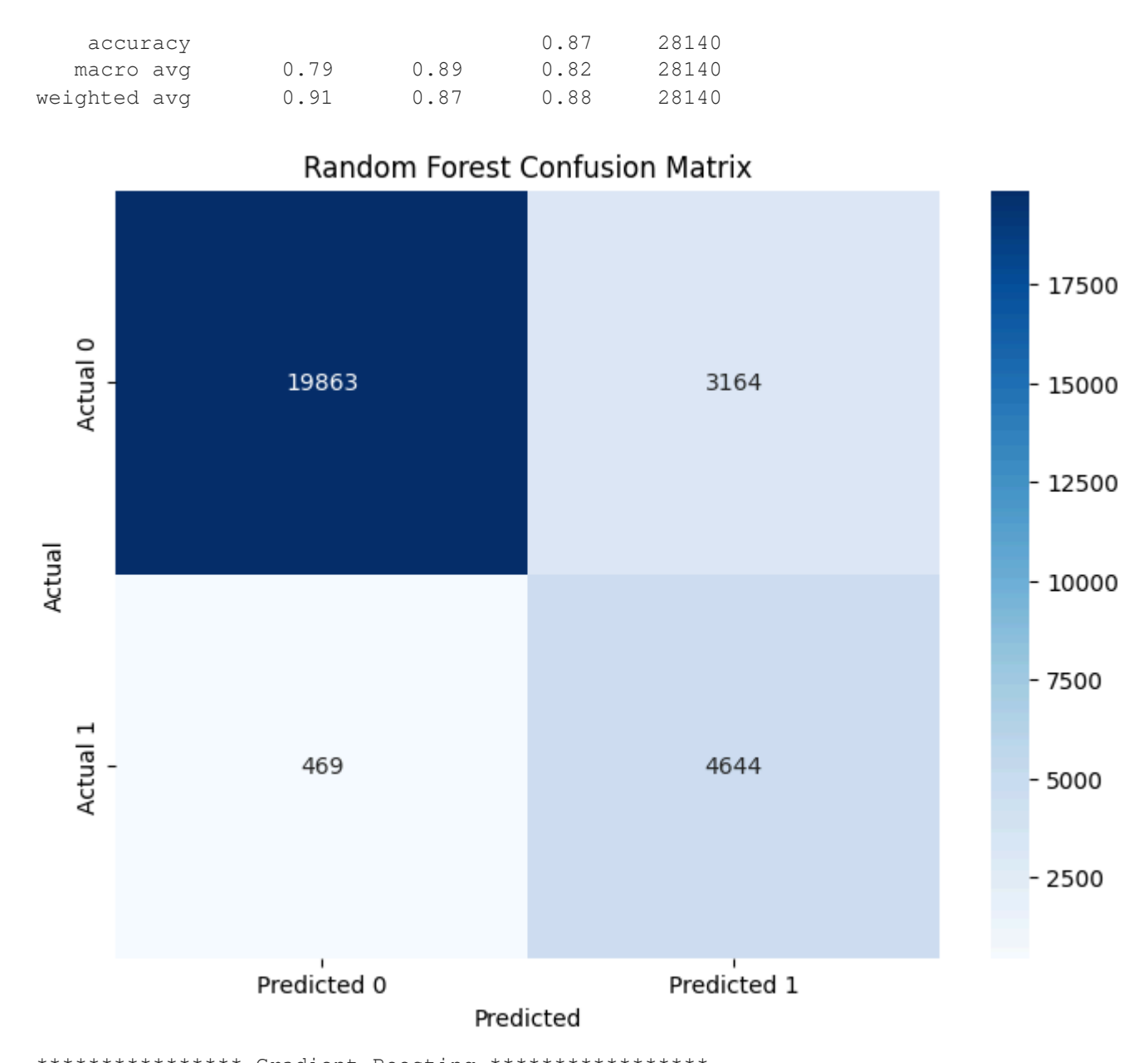
Definimos la variable *models* como un diccionario en el que se encuentran los tres modelos a comparar, esto hará que sea más cómodo a la hora de ajustar los modelos e interpretar los resultados

In [21]:
```
models = {
    'Decision Tree': DecisionTreeClassifier(max_depth=5, class_weight={0: 1, 1: 5}),
    'Random Forest': RandomForestClassifier(n_estimators=100, max_depth=5, n_jobs=10, class_weight={0: 1, 1: 5}),
    'Gradient Boosting': GradientBoostingClassifier(n_estimators=100, learning_rate=1.0, max_depth=1, random_state=0),
}

results = { 'Decision Tree': None, 'Random Forest': None, 'Gradient Boosting': None }

for name, model in models.items():
    results[name] = evaluate_model(model, X_train, y_train, X_test, y_test, name)
```

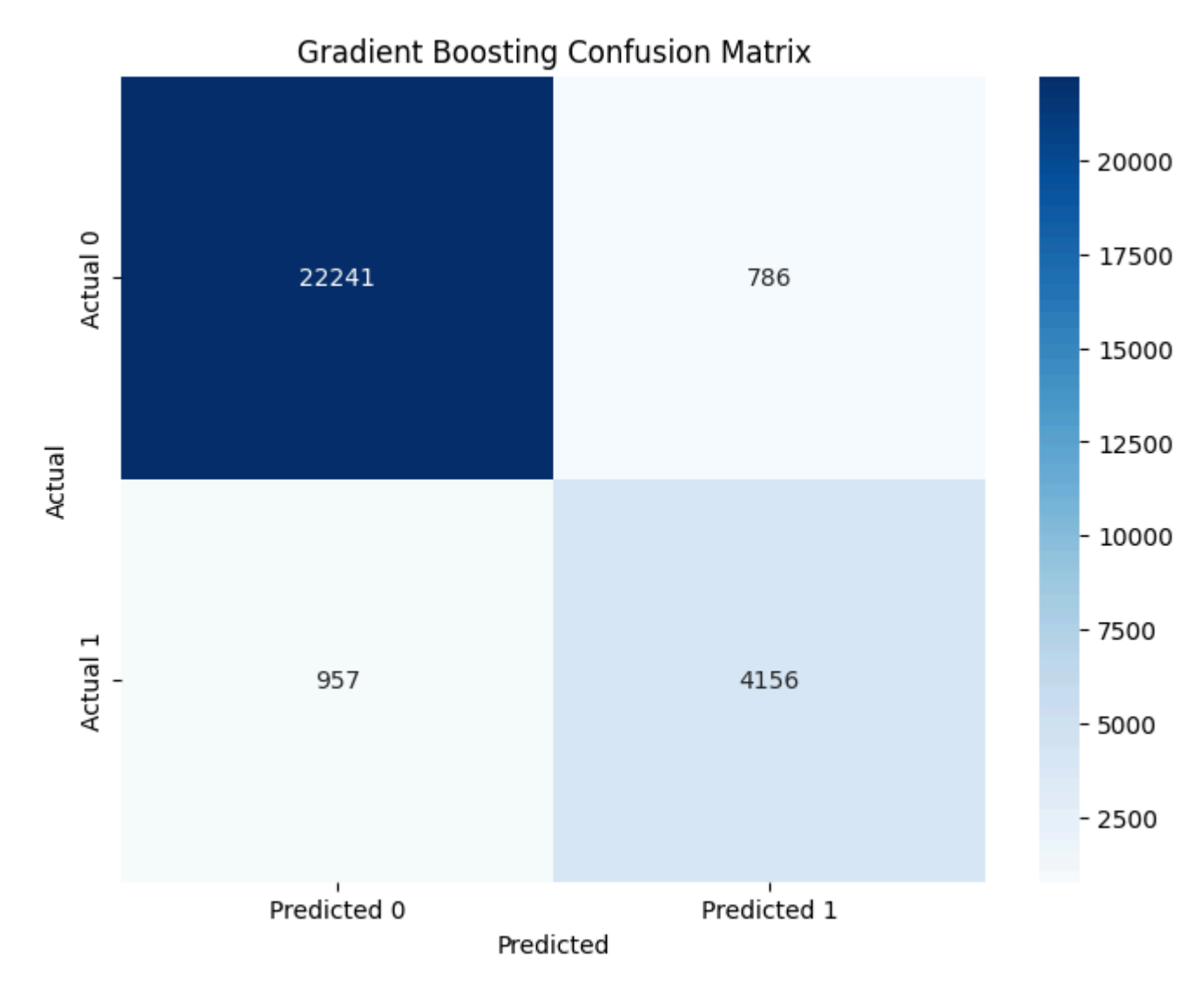
```
**************** Decision Tree *****************
Recall: 0.9268531194993155
--------------------------------------------
Classification Report 

              precision    recall  f1-score   support

           0       0.98      0.87      0.92     23027
           1       0.61      0.93      0.74      5113

    accuracy                           0.88     28140
   macro avg       0.80      0.90      0.83     28140
weighted avg       0.92      0.88      0.89     28140
```




```
**************** Random Forest *****************
Recall: 0.9082730295325641
--------------------------------------------
Classification Report 

              precision    recall  f1-score   support

           0       0.98      0.86      0.92     23027
           1       0.59      0.91      0.72      5113

    accuracy                           0.87     28140
   macro avg       0.79      0.89      0.82     28140
weighted avg       0.91      0.87      0.88     28140
```




```
**************** Gradient Boosting *****************
Recall: 0.8128300410717778
--------------------------------------------
Classification Report 

              precision    recall  f1-score   support

           0       0.96      0.97      0.96     23027
           1       0.84      0.81      0.83      5113

    accuracy                           0.94     28140
   macro avg       0.90      0.89      0.89     28140
weighted avg       0.94      0.94      0.94     28140
```



Conclusiones

A pesar de que tanto en los modelos Random Forest como Gradient Boosting se usan árboles de decisión, Sorprenedentemente el modelo que nos ofrece mejores resultados es el **Decission Tree** por sí solo

In [24]:
```
best_model = max(results.items(), key=lambda x: x[1][2])
print(f"Best model: {best_model[0]} with score: {best_model[1][2]}")
```

```
Best model: Decision Tree with score: 0.9268531194993155
```

Sin embargo, teóricamente ante problemas de clasificación como este en los que los datos están desbalanceados (el 82% pertenecen a una categoría y el 18% a otra) se presupone que **Gradient Boosting** es el modelo que debe funcionar mejor, pero en este caso ha sido todo lo contrario

In [24]:
```
worst_model = min(results.items(), key=lambda x: x[1][2])
print(f"Worst model: {worst_model[0]} with score: {worst_model[1][2]}")
```

```
Worst model: Gradient Boosting with score: 0.8128300410717778
```