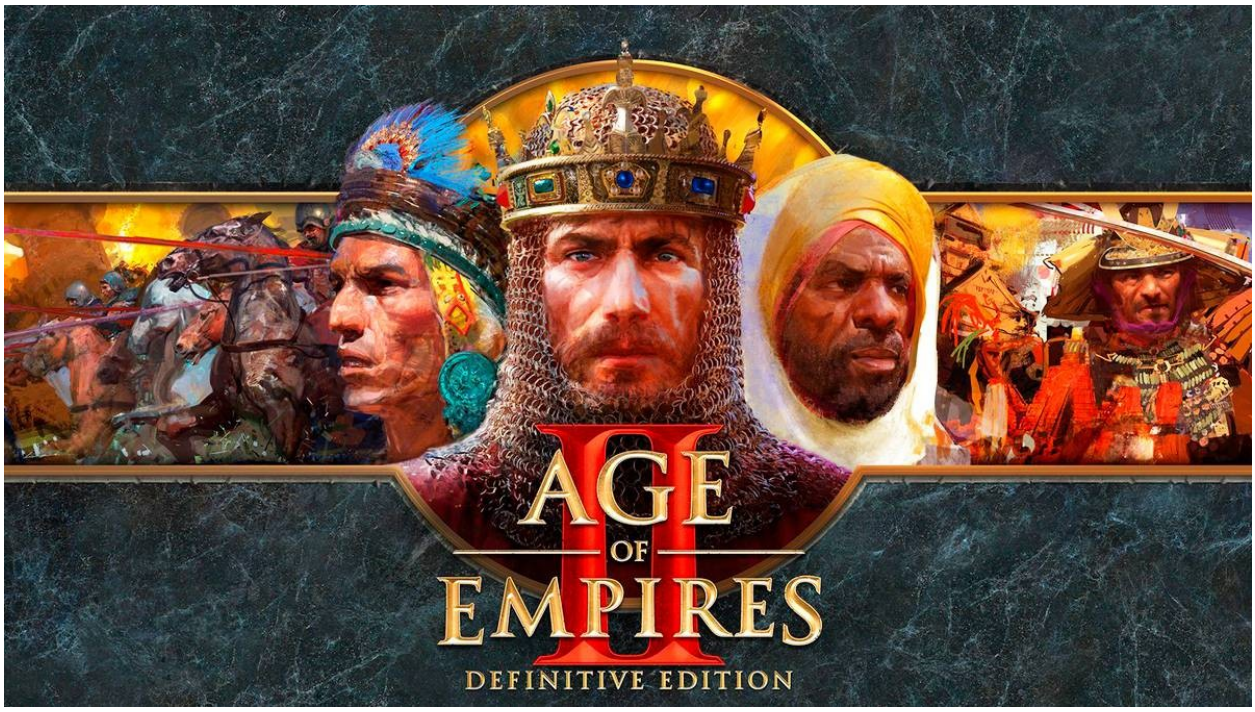
DuocUC® | ESCUELA DE INFORMÁTICA Y TELECOMUNICACIONES



Informe Técnico - Dataset de Age of Empires II

Integrante: Jonathan Huala

Docente: Giocrisrai Godoy

Sección: MLY0100 - 002D

Caso de Machine Learning

CONTEXTO CASO

Se analizará una Database del juego Age of Empires 2 para poder identificar diversas inquietudes y validarlas con los datos que se encuentran en esta, tales como si existen relaciones entre la victoria de una civilización entre otra en específico, ésto se hará verificando datos competitivos

del mismo juego y logrando diferenciar si hay alguna que tenga mayor ventaja o sea un "Counter" directo de otra. Se espera obtener los resultados por rangos del Ranking de los jugadores para identificar si se pueden seguir viendo eso entre otros rangos de jugadores y si hay una civilización con mayor porcentaje de ser jugada a diferencia del resto.

CASE CONTEXT

A database of the game Age of Empires 2 will be analyzed to identify various concerns and validate them with the data found within it, such as whether there are relationships between the victory of one civilization over another specifically. This will be done by verifying competitive data from the same game and determining if there is any civilization that has a greater advantage or is a direct "Counter" to another. The results by player ranking are expected to be obtained to identify if they can still be seen among other player rankings and if there is a civilization with a higher percentage of being played compared to the rest.

Dato	Descripción
$Token$	ID único de cada fila
Match	ID de cada partida
Rating	Rango al que pertenece el jugador al momento de jugarse la partida
Color	Color que el jugador eligió en la partida
Civ	Civilización que el jugador eligió en la partida
Team	Equipo del jugador
Winner	Indica si el jugador ganó o no la partida

Fase 1: Business Understanding

Preguntas

¿Qué civilización es la más jugada?

¿Hay alguna civilización que sea Counter de otra civilización? (Justificación: Clasificaremos como Counter que una civilización que sea jugada contra otra tenga un mayor winrate que la otra)

¿En los distintos rangos cambian los winrates de las civilizaciones?

Which civilization is the most played?

Is there any civilization that counters another civilization? (Justification: We will classify as Counter a civilization that, when played against another, has a higher win rate than the other)

Do the win rates of civilizations change at different ranks

Hipótesis

En este informe trabajaremos un máximo de tres hipótesis, la primera es que entre una y tres civilizaciones son las que más se juegan en general, entre todos los datos.

La segunda es que hay más de tres civilizaciones que tienen un mayor winrate contra otra civilización, por lo que sería un counter directo contra esta.

Y para finalizar que entre los diferentes rangos las civilizaciones tendrán una mayor o peor probabilidad de ganar contra su counter, a lo que queremos llegar es que entre los rangos 1000 y los 2000 los winrates de las civilizaciones no serán los mismos.

In this report, we will work with a maximum of three hypotheses; the first is that between one and three civilizations are the most played overall, among all the data.

The second is that there are more than three civilizations that have a higher win rate against another civilization, so it would be a direct counter to it.

And finally, that between the different ranks, civilizations will have a higher or lower probability of winning against their counter, what we want to convey is that between the ranks 1000 and 2000, the win rates of the civilizations will not be the same.

Criterios de aceptación

Para aceptar las hipótesis planteadas deben ocurrir los siguientes criterios (Serán enumerados por las hipótesis de este modo: 1. Civilización más jugada - 2. Civilización con counter - 3. Distintos winrates para civilizaciones dependiendo de rangos)

1.- Deberá haber una diferencia (Mínimo un 4%) entre unas civilizaciones a otras (Máximo 3 civilizaciones a diferencia del resto)

2.- Unas civilizaciones (Al menos 2) deben tener un mayor winrate (Al menos un 10% más) en partidas contra otra (Tener al menos un 60% de winrate contra otra civilización). (No es necesario que sean contra la misma, mientras sea contra otra civilización que no sea la evaluada se contará como si fuera un counter)

3.- Los winrate cambiarán (Al menos un 10% de diferencia) entre los distintos Ratings (Los ratings deben tener una diferencia de 800 más o menos).

To accept the proposed hypotheses, the following criteria must occur (They will be enumerated by the hypotheses as follows: 1. Most played civilization - 2. Civilization with counter - 3. Different win rates for civilizations depending on ranks)

1.- There must be a difference (at least 4%) between some civilizations and others (at most 3 civilizations different from the rest)

2.- Some civilizations (At least 2) must have a higher win rate (At least 10% more) in matches against another (Have at least a 60% win rate against another civilization). (It's not necessary for them to be against the same one; as long as it's against another civilization that is not the one being evaluated, it will count as a counter)

3.- The winrates will change (at least a 10% difference) between the different Ratings (the ratings must have a difference of about 800).

Fase 2: Data Understanding

Importaciones

```
import pandas as pd
import numpy as np
from sklearn.linear_model import ElasticNetCV, LogisticRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score, classification_report, accuracy_score, confusion_matrix,
roc_curve, roc_auc_score, precision_score, recall_score, f1_score
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import RandomForestRegressor,
RandomForestClassifier
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
```

Lectura del archivo

Se realiza la importación del archivo saltando las filas que tengan errores, esto dando 9.732.500 de filas aproximadamente

*Aclaración: Puede que muchos datos sean menores, esto se debe a que el detaset es muy grande y no está totalmente actualizado, por lo que al cargar los datos pueden ocurrir errores al cargar filas

The import of the file is carried out, skipping the rows that have errors, resulting in approximately 9,732,500 rows.

Clarification: Many data points may be smaller; this is because the dataset is very large and not fully updated, so errors may occur when loading rows.

Leer datos en kedro

```
aoe = pd.read_csv('../data/01_raw/match_players.csv',
on_bad_lines='skip')
aoe
```

```
---------------------------------------------------------------------
-----
FileNotFoundError                         Traceback (most recent call
last)
<ipython-input-2-000ea6fb3a89> in <cell line: 1>()
----> 1 aoe = pd.read_csv('../data/01_raw/match_players.csv',
on_bad_lines='skip')
      2 aoe

/usr/local/lib/python3.10/dist-packages/pandas/io/parsers/readers.py
```

```
in read_csv(filepath_or_buffer, sep, delimiter, header, names,
index_col, usecols, dtype, engine, converters, true_values,
false_values, skipinitialspace, skiprows, skipfooter, nrows,
na_values, keep_default_na, na_filter, verbose, skip_blank_lines,
parse_dates, infer_datetime_format, keep_date_col, date_parser,
date_format, dayfirst, cache_dates, iterator, chunksize, compression,
thousands, decimal, lineterminator, quotechar, quoting, doublequote,
escapechar, comment, encoding, encoding_errors, dialect, on_bad_lines,
delim_whitespace, low_memory, memory_map, float_precision,
storage_options, dtype_backend)
   1024     kwds.update(kwds_defaults)
   1025
-> 1026     return _read(filepath_or_buffer, kwds)
   1027
   1028

/usr/local/lib/python3.10/dist-packages/pandas/io/parsers/readers.py
in _read(filepath_or_buffer, kwds)
    618
    619     # Create the parser.
--> 620     parser = TextFileReader(filepath_or_buffer, **kwds)
    621
    622     if chunksize or iterator:

/usr/local/lib/python3.10/dist-packages/pandas/io/parsers/readers.py
in __init__(self, f, engine, **kwds)
   1618
   1619         self.handles: IOHandles | None = None
-> 1620         self._engine = self._make_engine(f, self.engine)
   1621
   1622     def close(self) -> None:

/usr/local/lib/python3.10/dist-packages/pandas/io/parsers/readers.py
in _make_engine(self, f, engine)
   1878                 if "b" not in mode:
   1879                     mode += "b"
-> 1880             self.handles = get_handle(
   1881                 f,
   1882                 mode,

/usr/local/lib/python3.10/dist-packages/pandas/io/common.py in
get_handle(path_or_buf, mode, encoding, compression, memory_map,
is_text, errors, storage_options)
    871         if ioargs.encoding and "b" not in ioargs.mode:
    872             # Encoding
--> 873             handle = open(
    874                 handle,
    875                 ioargs.mode,
```

```
FileNotFoundError: [Errno 2] No such file or directory:
'../data/01_raw/match_players.csv'
```

Leer datos en Google Colab

```
aoe = pd.read_csv('match_players.csv', on_bad_lines='skip')
aoe
```

```
<ipython-input-3-1274c3708521>:1: DtypeWarning: Columns (6) have mixed
types. Specify dtype option on import or set low_memory=False.
  aoe = pd.read_csv('match_players.csv', on_bad_lines='skip')
```

{"type":"dataframe","variable_name":"aoe"}

Exploración de datos

Identificar que tipos de datos estamos trabajando para saber cómo manejarlos

Identify what types of data we are working with to know how to handle them.

```
print(aoe.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5287633 entries, 0 to 5287632
Data columns (total 7 columns):
 #   Column  Dtype
---  ------  -----
 0   token   object
 1   match   object
 2   rating  float64
 3   color   object
 4   civ     object
 5   team    float64
 6   winner  object
dtypes: float64(2), object(5)
memory usage: 282.4+ MB
None
```

Identificación de filas con valores nulos

Identification of rows with null values

```
valores_nulos = aoe.isnull().any(axis=1)

print("Filas con valores nulos:")
print(aoe[valores_nulos])
```

```
Filas con valores nulos:
                    token             match  rating   color
civ  team  \
```

```
120      spacWUMyisaiQWbI  oUsF7ZFMF0ykgCHm     NaN     Red
Celts   1.0
289      PLQJ3MtJgZwNEOf4  N6Hei3RaxUw0Ij2e     NaN  Yellow
Huns   2.0
336      u3UVup3DX9ng1ktx  L5IlyPsOnDJcfqL6     NaN  Orange
Celts   1.0
344      agqEXgU4ClQx3zdW  zNv6NyWyaax5qACt     NaN  Yellow
Mayans   1.0
435      j9DHxfJcpMyAfhY2  em7f5lJ0KaD36W8a     NaN    Blue
Franks   2.0
...                   ...               ...     ...     ...          ..
.   ...
5287368  ElwFMD935OXy3eBj  4avBXoZnyJF4OLOG     NaN  Yellow
Franks   1.0
5287388  HXO8EzBqdDH9GFVC  BOMD8OGh0yfaunWe     NaN  Orange
Berbers   1.0
5287390  ijqY4PZSkPNIaBhr  BOMD8OGh0yfaunWe     NaN     Red
Magyars   1.0
5287432  J8GGHyxH8joASQyz  5oDadt2ndoA7rm6s     NaN  Purple
Vietnamese   1.0
5287632  oGgwr6n05HQTol9T  IYEkxISrc6FY4c80  1098.0       Y
NaN   NaN

        winner
120       True
289       True
336      False
344       True
435       True
...        ...
5287368   True
5287388  False
5287390  False
5287432   True
5287632    NaN

[273686 rows x 7 columns]
```

Los datos nulos son un aproximado de 503.445, que sería un 19,33% aproximadamente.

The null data is approximately 503,445, which would be around 19.33%.

Se completarán y/o eliminaran los datos nulos en la fase 3, pero antes de eso se debe sacar el promedio de rating por cada match para llenar el valor de esas filas faltantes [Fase 3]

Null data will be completed and/or deleted in phase 3, but before that, the average rating per match must be calculated to fill in the missing values for those rows [Phase 3]

Antes de sacar el promedio se deben pasar los datos de rating a entero, esto se hará en la fase 3 ("RATING A ENTERO")

Before calculating the average, the rating data must be converted to integers; this will be done in phase 3. ("RATING A ENTERO")

```
promedio_rating_match = aoe.groupby('match')['rating'].mean()
```

*Luego de esto se llenarán los datos nulos (Llenado de datos nulos) [Fase 3]

*After this, the null data will be filled in (Null Data Filling) [Phase 3]

VERIFICAR VALORES NULOS RATING

```
print(aoe['rating'].isnull().sum())
```

```
2
```

Crear una nueva partición, eliminar filas con valores nulos en Match y examinar las filas con valores nulos restantes

Create a new partition, remove rows with null values in Match, and examine the remaining rows with null values.

*Esto se hace por si a futuro se pueden llenar los datos de la columna Match ocupando un modelo de regresión o clasificación.

*This is done in case the data in the Match column can be filled in the future using a regression or classification model.

```
aoe_2 = aoe.dropna(subset=['match'])

print(aoe_2.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5287633 entries, 0 to 5287632
Data columns (total 7 columns):
 #   Column  Dtype
---  ------  -----
```

```
 0   token   object
 1   match   object
 2   rating  float64
 3   color   object
 4   civ     object
 5   team    float64
 6   winner  object
dtypes: float64(2), object(5)
memory usage: 282.4+ MB
None
```

Ahora identificar si contiene datos nulos en alguna fila y cuantas filas tienen nuestros datos

Now identify if it contains null data in any row and how many rows our data has.

```
filas_con_nulos = aoe_2[aoe_2.isnull().any(axis=1)]

print("Filas con datos nulos en 'aoe_2':")
print(filas_con_nulos)
```

```
Filas con datos nulos en 'aoe_2':
                  token             match  rating color      civ
team  \
2516081  ugTmJlyQ4h46qIin  f8049f6242ef95b1   913.0   NaN   Aztecs
1.0
5286165  bM7s4c5dOLcbrJXU  4aa8c22e8b931f68     NaN  Blue  Britons
2.0
5286556  GqLfJK8hN82nPItb  9495a709fc746d8a     NaN   Red  Vikings
1.0
5287632  oGgwr6n05HQTol9T  IYEkxISrc6FY4c80  1098.0     Y      NaN
NaN

        winner
2516081  False
5286165   True
5286556  False
5287632    NaN
```

*Los datos nulos son el 0.002% aproximadamente de los datos totales y son dificilmente recuperables ya que tienen 2 o más datos NaN o datos NaN que no tienen relación con el resto, por lo cual se eliminarán de los datos que ocuparemos. [Esto se hará en la fase 3 (Eliminación de los otros NaN)] [Fase 3]

*The null data is approximately 0.002% of the total data and is hardly recoverable since it has 2 or more NaN values or NaN values that are not related to the rest, so they will be removed from the data we will use. [This will be done in phase 3 (Removal of other NaNs)] [Phase 3]

ÚLTIMA VERIFICACIÓN NULOS

```
print(aoe_2[aoe_2.isnull().any(axis=1)])
```

```
Empty DataFrame
Columns: [token, match, rating, color, civ, team, winner]
Index: []
```

*Ahora que todos los valores nulos fueron eliminados podremos trabajar en nuestro dataset

*Now that all the null values have been removed, we can work on our dataset.

Exploración de datos en la columna de Rating

*Se realizará una exploración de los datos de la columna Rating, ya que estos serán ocupados para verificar si cambia el uso de las civilización en los diferentes Ratings y verificar sus winrates.

*An exploration of the data in the Rating column will be conducted, as this will be used to verify if the use of civilizations changes across different Ratings and to check their win rates.

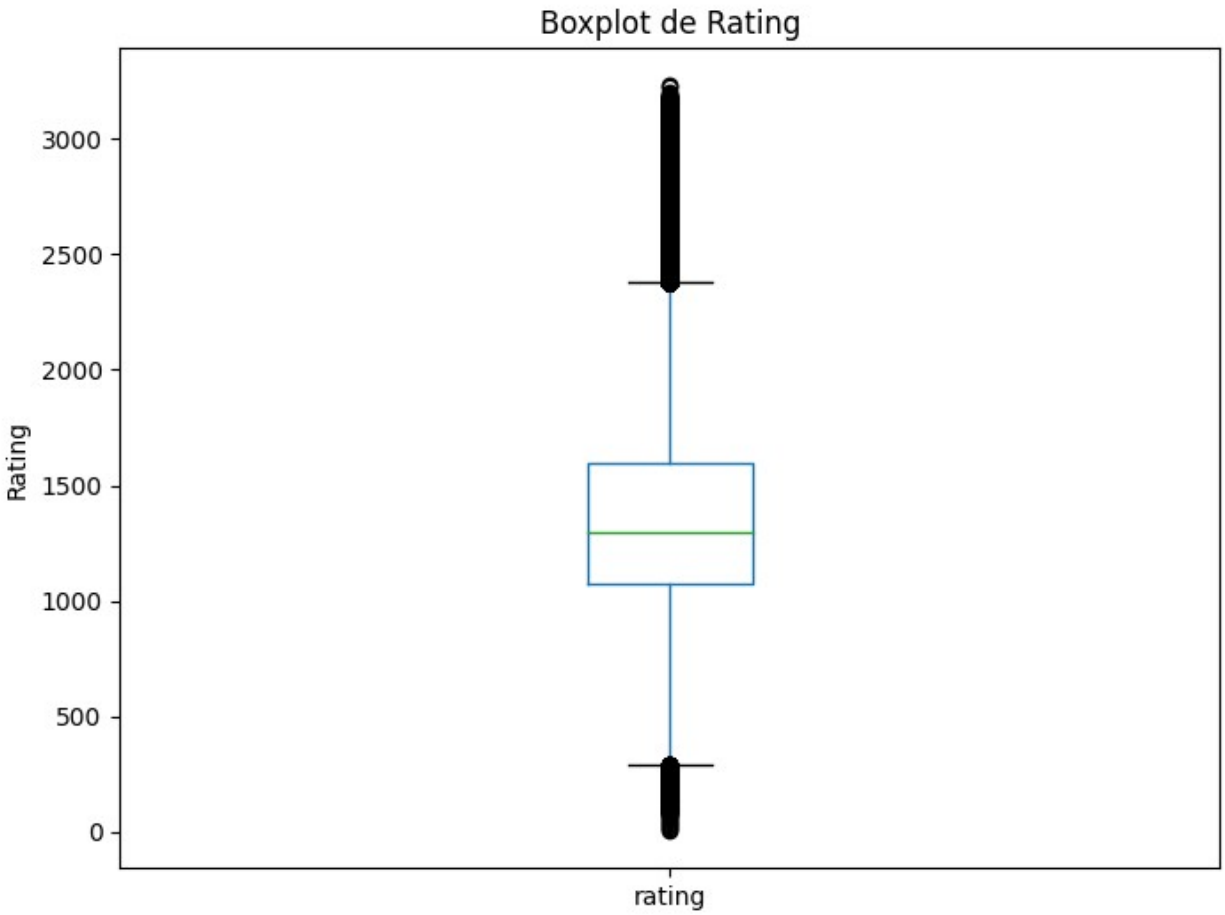
```
print(aoe_2['rating'].describe())
```

```
count    5.287629e+06
mean     1.377153e+03
std      4.164944e+02
min      4.000000e+00
25%      1.073000e+03
50%      1.299000e+03
75%      1.596000e+03
max      3.231000e+03
Name: rating, dtype: float64
```

Aquí podemos identificar unas anomalías, ya que el mínimo que se muestra en el rating es muy bajo y el máximo es mucho más alto que la media, esto se analizará más a profundidad en el boxplot para identificar los valores atipicos.

Here we can identify some anomalies, as the minimum shown in the rating is very low and the maximum is much higher than the average. This will be analyzed in more depth in the boxplot to identify the outliers.

```
plt.figure(figsize=(8, 6))
aoe_2.boxplot(column='rating')
plt.title('Boxplot de Rating')
plt.ylabel('Rating')
plt.grid(False)
plt.show()
```



Como se especificó en la exploración de rating con los datos estadísticos se puede ver la gran anormalidad de los datos mencionados, estos datos atipicos se deberían eliminar ya que pueden ser errores los cuales puedan afectar el resto de datos.

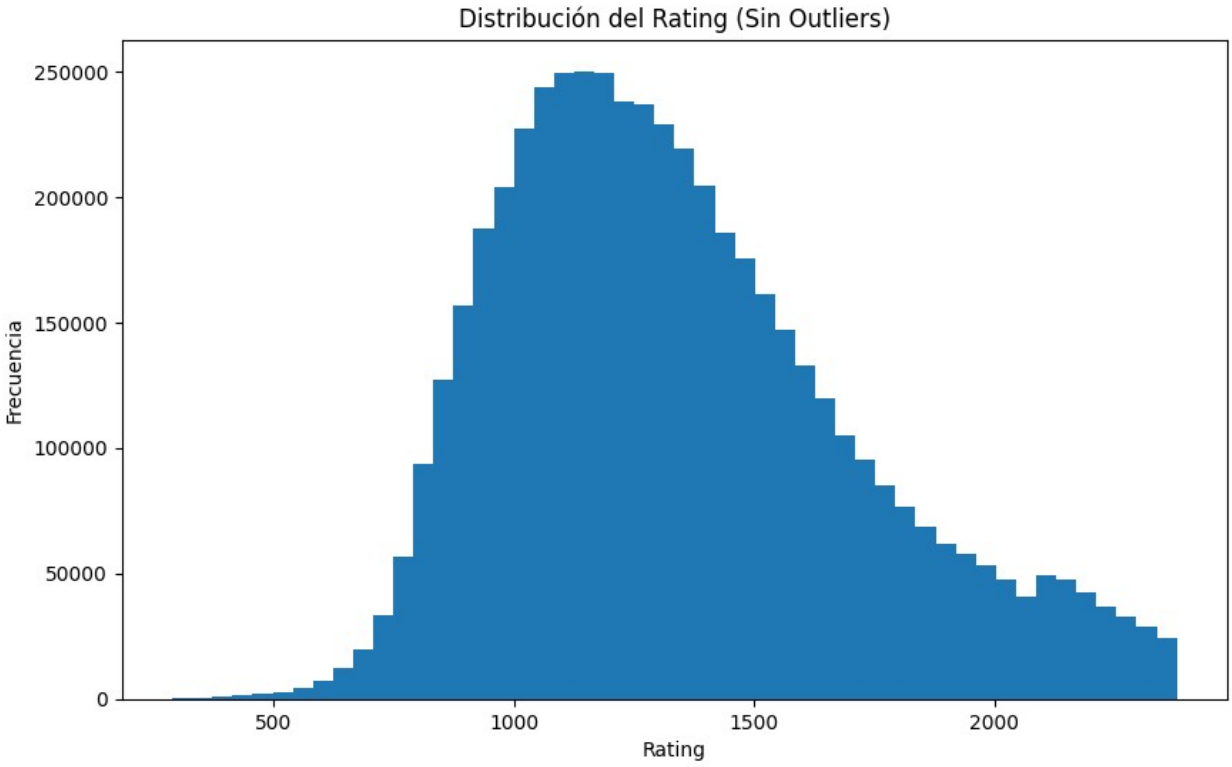
Aquí sacaremos los cuartiles y el rango intercuartil para luego realizar una limpieza de los outliers, también se realizará otra partición por temas de seguridad.

As specified in the rating exploration with the statistical data, the great abnormality of the mentioned data can be seen; these outlier data should be removed as they may be errors that could affect the rest of the data.

Here we will calculate the quartiles and the interquartile range to then clean the outliers, and another partition will be made for security reasons.

```
Q1_rating = aoe_2['rating'].quantile(0.25)
Q3_rating = aoe_2['rating'].quantile(0.75)
IQR_rating = Q3_rating - Q1_rating

lower_bound_rating = Q1_rating - 1.5 * IQR_rating
upper_bound_rating = Q3_rating + 1.5 * IQR_rating
```

```
aoe_2_sin_outliers = aoe_2[(aoe_2['rating'] >= lower_bound_rating) &
(aoe_2['rating'] <= upper_bound_rating)]

plt.figure(figsize=(10, 6))
aoe_2_sin_outliers['rating'].hist(bins=50)
plt.title('Distribución del Rating (Sin Outliers)')
plt.xlabel('Rating')
plt.ylabel('Frecuencia')
plt.grid(False)
plt.show()
```



*Como se puede apreciar los datos se encuentran más concentrados para el manejo de estos, los datos que podemos manejar luego de eliminar los outliers.

Verificamos si los datos en Winner es True o False

*As can be seen, the data is more concentrated for handling, the data we can manage after removing the outliers.

We check if the data in Winner is True or False.

```
valores_unicos = aoe_2_sin_outliers['winner'].unique()
print("Valores únicos en la columna 'winner':")
print(valores_unicos)
```

```
Valores únicos en la columna 'winner':
[False True]
```

Ahora trabajaremos con la columna win, la transformaremos a enteros [True=1, False=0] para poder manejar el winrate, el cual es el porcentaje de que ocurra una victoria o una derrota, éste se encuentra entre 0 y 100. (TRANSFORMACIÓN DE WINNER) [Fase 3]

Now we will work with the win column, we will transform it to integers [True=1, False=0] to be able to handle the winrate, which is the percentage of a victory or a defeat occurring, it ranges between 0 and 100. (WINNER TRANSFORMATION) [Phase 3]

¿Qué civilización es la más jugada?

Para responder esa pregunta realizaremos el conteo de cada civilización y luego lo graficaremos.

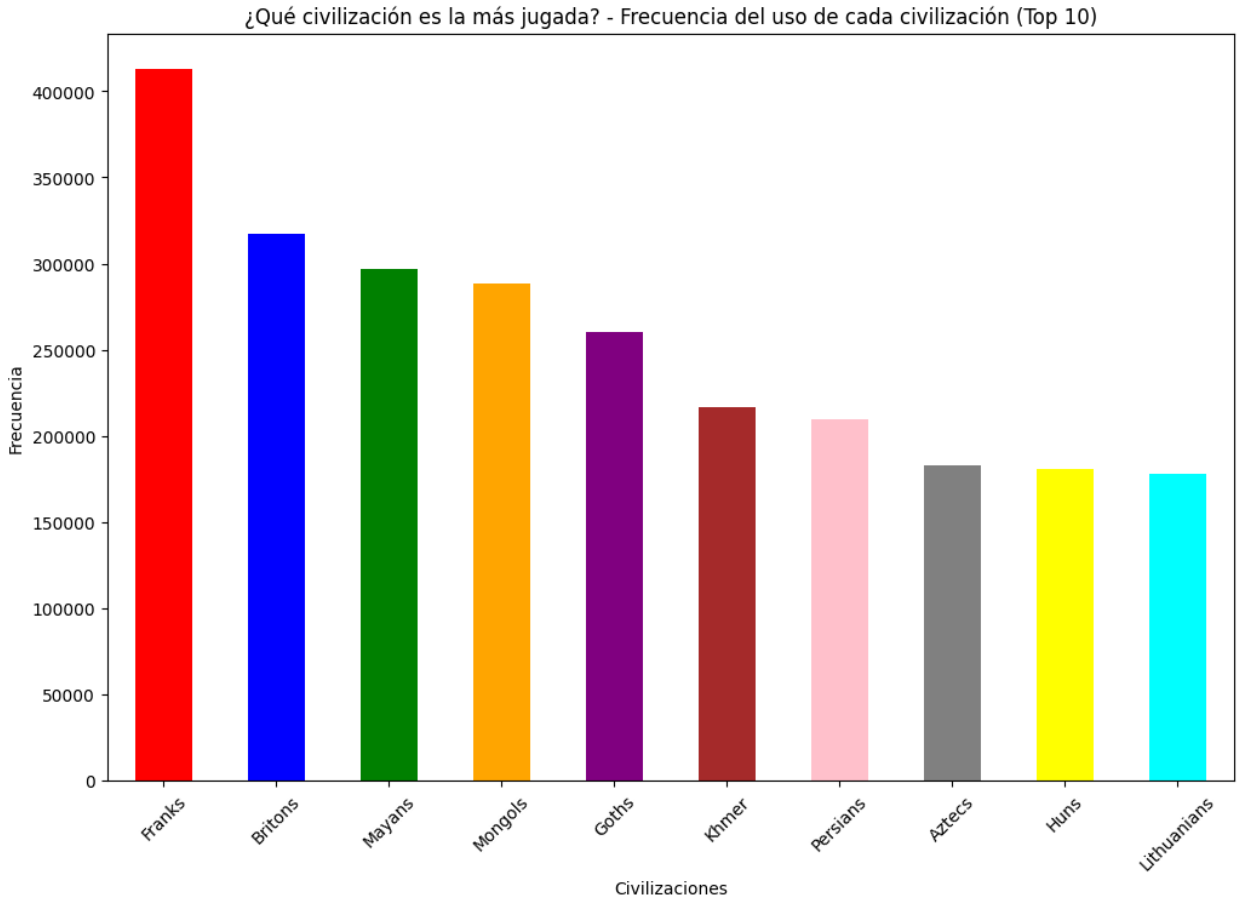
To answer that question, we will count each civilization and then graph it.

```
conteo_de_civilizaciones = aoe_2_sin_outliers['civ'].value_counts()
```

Gráfico de las civilizaciones y el uso de éstas en el dataset

Graph of civilizations and their use in the dataset

```
plt.figure(figsize=(12, 8))

top_10_civilizaciones = conteo_de_civilizaciones.head(10)
colores = ['red', 'blue', 'green', 'orange', 'purple', 'brown',
'pink', 'gray', 'yellow', 'cyan']
top_10_civilizaciones.plot(kind='bar', color=colores)

plt.title('¿Qué civilización es la más jugada? - Frecuencia del uso de
cada civilización (Top 10)')
plt.xlabel('Civilizaciones')
plt.ylabel('Frecuencia')
plt.xticks(rotation=45)
plt.show()
```



Calculo de la diferencia entre la primera civilización más jugada y la segunda.

Calculation of the difference between the most played first civilization and the second.

```
franks_britons =
aoe_2_sin_outliers[aoe_2_sin_outliers['civ'].isin(['Franks',
'Britons'])]


conteo_franks_britons = franks_britons['civ'].value_counts()


diferencia_conteo = conteo_franks_britons['Franks'] -
conteo_franks_britons['Britons']
porcentaje_diferencia = (diferencia_conteo /
conteo_franks_britons['Franks']) * 100

print(f"Diferencia entre la primera civilización más jugada y la
segunda: {porcentaje_diferencia:.2f}% ({diferencia_conteo} partidas)")
```

```
Diferencia entre la primera civilización más jugada y la segunda:
23.07% (95203 partidas)
```

Viendo el gráfico y los datos obtenidos hay una gran diferencia entra las dos civilizaciones, se podría decir que la primera hipótesis es correcta, la diferencia entre la primera y la segunda es del 23.12%, lo cual hace a los 'Franks' la civilización más jugada y si hay una civilización más jugada

Looking at the graph and the obtained data, there is a significant difference between the two civilizations. It could be said that the first hypothesis is correct; the difference between the first and the second is 23.12%, which makes the 'Franks' the most played civilization, and if there is a more played civilization.

¿Hay alguna civilización que sea Counter de otra civilización?

Parte 1

Colores que se usarán para los gráficos

Colors that will be used for the charts

```
colores_civilizaciones = {
    'Franks': 'red',
    'Britons': 'blue',
    'Mayans': 'green',
    'Mongols': 'orange',
    'Goths': 'purple',
    'Khmer': 'brown',
    'Persians': 'pink',
    'Aztecs': 'gray',
    'Huns': 'yellow',
    'Lithuanians': 'cyan',
    'Indians': 'pink',
    'Malians': 'black',
    'Slavs': 'slateblue',
    'Incas': 'coral',
    'Magyars': 'orchid',
    'Koreans': 'mistyrose',
    'Teutons': 'lime',
    'Portuguese': 'steelblue',
    'Berbers': 'darkseagreen',
    'Burmese': 'slateblue',
  }
```

Para comenzar a resolver esta pregunta se realizará un conteo de los matches, al igual que se dejarán los matches donde solo hay dos jugadores, ya que si hay uno solo o más de dos no se podrá validar un counter directo contra otra civilización, ya que pueden influir más factores o si en la partida es una persona ningúna otra civilización sería evaluada, luego de dejar solo esos datos se hará una partición donde solo quedarán los datos válidos para responder esta pregunta

To begin solving this question, a count of the matches will be conducted, and matches with only two players will be retained, as if there is only one or more than two, a direct counter against

another civilization cannot be validated, since more factors may influence or if in the match there is only one person, no other civilization would be evaluated. After retaining only those data, a partition will be made where only the valid data to answer this question will remain.

```
conteo_matches = aoe_2_sin_outliers['match'].value_counts()

matches_validos = conteo_matches[conteo_matches == 2].index

aoe_2_h2 =
aoe_2_sin_outliers[aoe_2_sin_outliers['match'].isin(matches_validos)]
```

A continuación se realizarán varios pasos para poder pasar a la respuesta de la pregunta, primero se ordenará el dataframe por match y luego por winner, luego se filtrarán los datos, luego se crearán las columnas civilización_win y civilización_lose, luego se realiza una unión entre las columnas match, promedio_rating (Promedio del rating de la partida), civilización_win y civilización_lose y se reenombran. Esta partición de los datos se realiza para poder trabajar con la columna match, la civilización que perdió la partida y la que ganó, para poder sacar un conteo de partidas ganadas contra cada civilización y obtener un winrate contra cada una.

*Aclaración: Éste código dura 4min aproximadamente y uno realizado con anterioridad tardaba en ejecutarse y dar los resultados 48min.

The following steps will be taken to arrive at the answer to the question: first, the dataframe will be sorted by match and then by winner, then the data will be filtered, then the columns civilization_win and civilization_lose will be created, and finally, a join will be performed between the columns match, average_rating (average rating of the match), civilization_win, and civilization_lose, and they will be renamed. This partitioning of the data is done to work with the match column, the civilization that lost the game and the one that won, in order to get a count of games won against each civilization and obtain a win rate against each one.

Clarification: This code takes approximately 4 minutes, while one done previously took 48 minutes to execute and provide results.

```
aoe_2_h2_sorted = aoe_2_h2.sort_values(by=['match', 'winner'])
matches_filtrados2 = aoe_2_h2_sorted.groupby('match').filter(lambda x:
len(x) == 2 and set(x['winner']) == {0, 1})

civilizacion_win = matches_filtrados2.loc[matches_filtrados2['winner']
== 1, ['match', 'civ']]
civilizacion_lose =
matches_filtrados2.loc[matches_filtrados2['winner'] == 0, ['match',
'civ']]

promedio_rating = matches_filtrados2.groupby('match')
['rating'].mean().reset_index()

particion_final = pd.merge(civilizacion_win, civilizacion_lose,
on='match', suffixes=('_win', '_lose'))
particion_final = pd.merge(particion_final, promedio_rating,
on='match')
```

```
particion_final.columns = ['match', 'civilizacion_win',
'civilizacion_lose', 'promedio_rating']
```

Particiones para los modelos de clasificación y regresión

Para el modelo de regresión se realizará la predicción de que winrate tiene al enfrentamarse contra una civilización, para esto se espera ocupar los datos en civilizacion_win y civilizacion_lose, donde se ocupará la civilización que ocupará la persona y la civilización que ocupará el oponente, así obtener un estimado de que tan probable es que gane o pierda.

For the regression model, the prediction of the winrate when facing a civilization will be made. For this, it is expected to use the data in civilization_win and civilization_lose, where the civilization the person will use and the civilization the opponent will use will be taken into account, thus obtaining an estimate of how likely it is to win or lose.

```
modelo_regresion_datos1 = particion_final
```

Para el modelo de clasificación se espera poder predecir contra que civilizaciones una civilización elegida puede perder y contra cuales puede ganar, para esto se espera ocupar arreglos y obtener el porcentaje de victoria o derrota que pueda obtener la civilización, en su defecto se obtendrá contra que civilización tiene mayor probabilidad de ganar en un rango específico

Para el modelo de clasificación se espera poder predecir contra que civilizaciones una civilización elegida puede perder y contra cuales puede ganar, para esto se espera ocupar arreglos y obtener el porcentaje de victoria o derrota que pueda obtener la civilización, en su defecto se obtendrá contra que civilización tiene mayor probabilidad de ganar en un rango específico

```
modelo_clasificacion_datos1 = particion_final
```

Parte 2

Para continuar y ya finalizando se realizará un conteo de las victorias, derrotas y la diferencia entre éstas para cada combate entre civilizaciones, para así poder obtener un resultado a que hay un counter para cada civilización.

Se crea una lista para almacenar los resultados finales y se obtienen todas las combinaciones únicas de cada civilizaciones, luego con cíclicos y condicionales se realiza una exploración por todos los datos para calcular las victorias y derrotas contra cada oponentes, ésto se realiza con dos ciclos for y un if, donde primero se calculan las victorias de cada civilización contra su oponente y las derrotas del mismo, luego se calcula la diferencia entre las victorias y derrotas y se agregan los resultados a la lista creada al comienzo.

Al finalizar el ciclo for principal se transforma la lista en un dataframe y se ordenan de forma descendente en la columna diferencia.

*Aclaración: Éste código tarda en ejecutarse aproximadamente 15min

To continue and wrap up, a count of the victories, defeats, and the difference between them for each battle between civilizations will be conducted, in order to obtain a result indicating that there is a counter for each civilization.

A list is created to store the final results, and all unique combinations of each civilization are obtained. Then, using loops and conditionals, an exploration of all the data is carried out to calculate the victories and defeats against each opponent. This is done with two for loops and an if statement, where first the victories of each civilization against their opponent and the defeats of the same are calculated. Then, the difference between the victories and defeats is calculated, and the results are added to the list created at the beginning.

At the end of the main for loop, the list is transformed into a dataframe and sorted in descending order by the difference column.

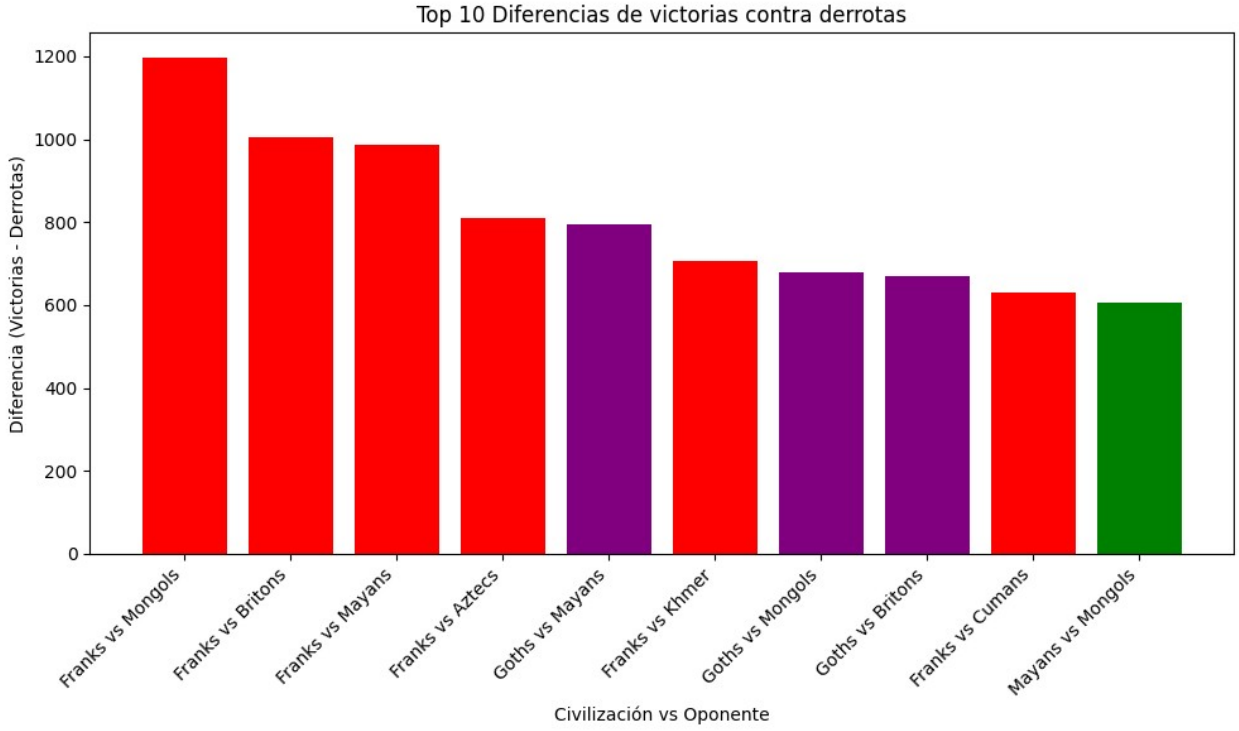
Clarification: This code takes approximately 15 minutes to execute.

```
resultados_finales = []
civilizaciones = pd.concat([particion_final['civilizacion_win'],
particion_final['civilizacion_lose']]).unique()

for civ in civilizaciones:
    for oponente in civilizaciones:
        if civ != oponente:
            victorias = particion_final[
                (particion_final['civilizacion_win'] == civ) &
(particion_final['civilizacion_lose'] == oponente)
            ].shape[0]
            derrotas = particion_final[
                (particion_final['civilizacion_win'] == oponente) &
(particion_final['civilizacion_lose'] == civ)
            ].shape[0]
            diferencia = victorias - derrotas
            resultados_finales.append({
                'civilizacion': civ,
                'oponente': oponente,
                'victorias': victorias,
                'derrotas': derrotas,
                'diferencia': diferencia
            })

resultados_finales_df = pd.DataFrame(resultados_finales)
resultados_finales_df =
resultados_finales_df.sort_values(by=['diferencia'], ascending=False)
```

Ahora pasaremos a mostrar el gráfico sobre la diferencias entre las victorias y las derrotas entre las civilizaciones, siendo la primera la que tiene la victoria y la segunda que se muestra teniendo la derrota. para esto se ocuparán 10 datos para mostrar los más relevantes y poder leer las civilizaciones correctamente

Now we will show the graph about the differences between victories and defeats among civilizations, with the first being the one that has the victory and the second being the one that shows the defeat. For this, 10 pieces of data will be used to show the most relevant ones and be able to read the civilizations correctly.

```
top_50_resultados = resultados_finales_df.head(10)

colores_barras =
top_50_resultados['civilizacion'].map(colores_civilizaciones)
plt.figure(figsize=(10, 6))
plt.bar(top_50_resultados['civilizacion'] + " vs " +
top_50_resultados['oponente'],
        top_50_resultados['diferencia'], color=colores_barras)

plt.xticks(rotation=45, ha='right')
plt.xlabel('Civilización vs Oponente')
plt.ylabel('Diferencia (Victorias - Derrotas)')
plt.title('Top 10 Diferencias de victorias contra derrotas')
plt.tight_layout()
plt.show()
```

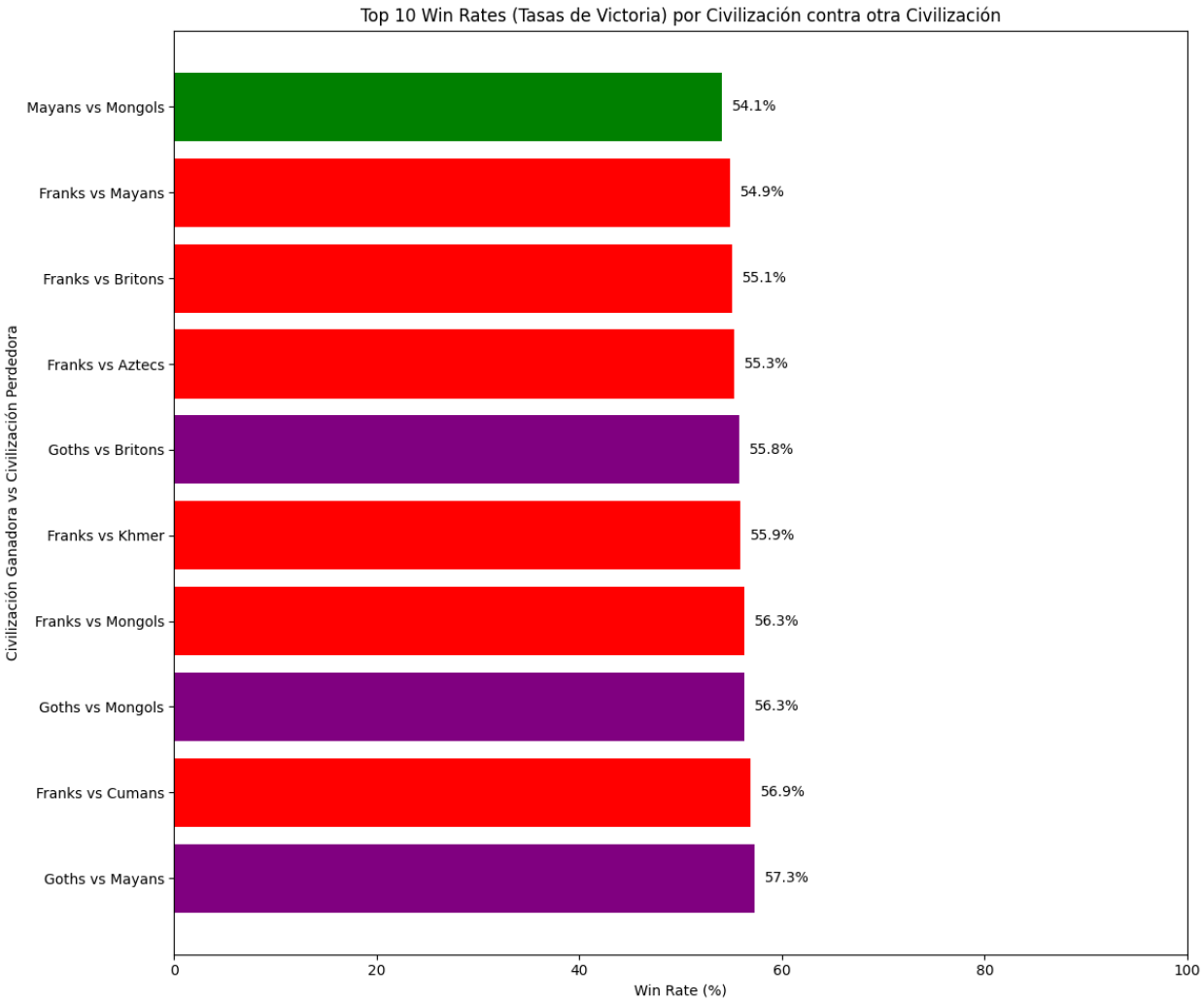


Para finalizar sacaremos el winrate y lo graficaremos, para saber que tanta probabilidad tendrá de ganar cada civilización que se enfrente a otra entre los 10 mejores y los 10 peores enfrentamientos entre civilizaciones.

```
top_50_resultados['winrate_percent'] = (top_50_resultados['victorias']
/

(top_50_resultados['victorias'] + top_50_resultados['derrotas'])) *
100

top_50_resultados =
```

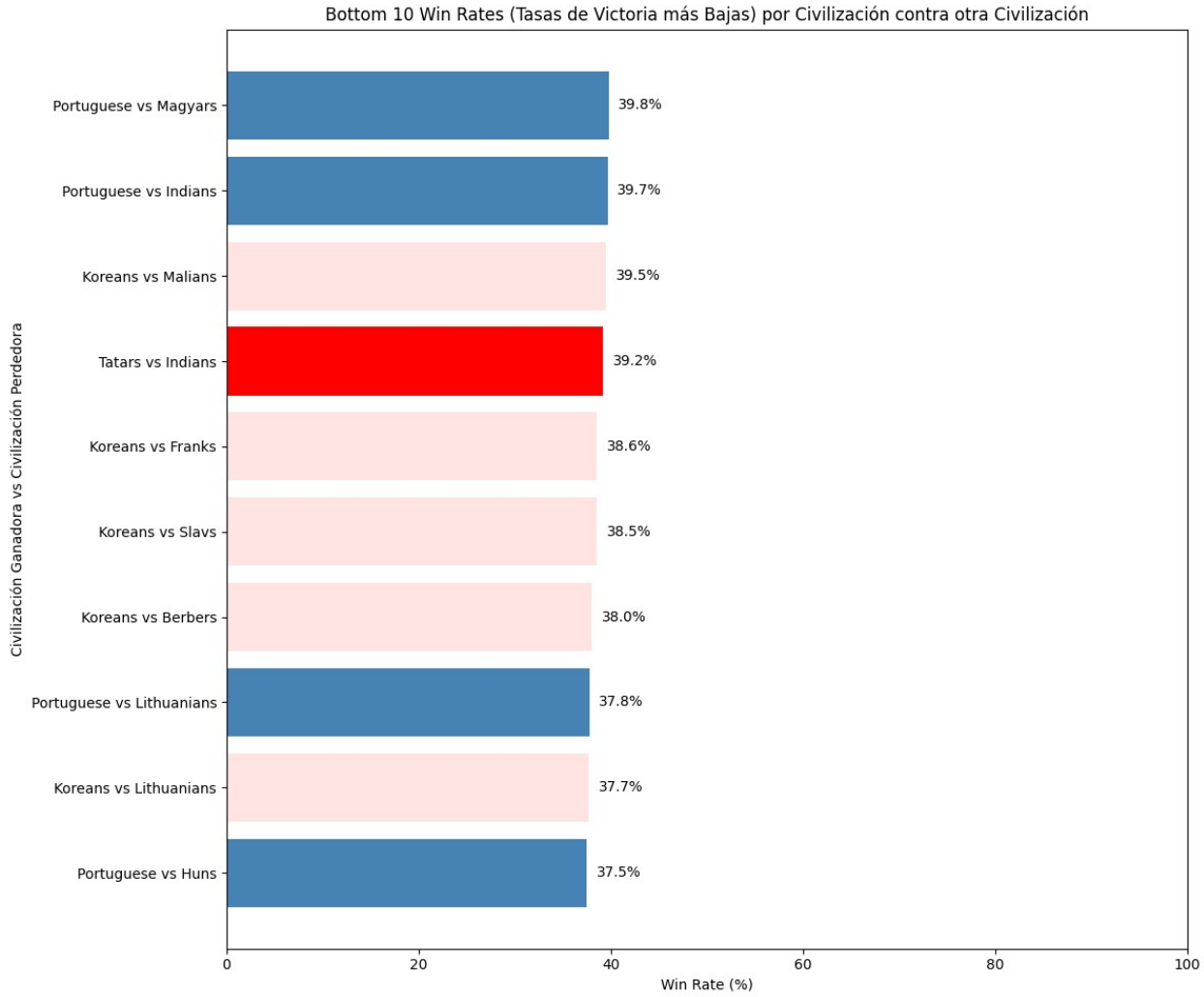
```
top_50_resultados.sort_values(by='winrate_percent', ascending=False)

plt.figure(figsize=(12, 10))
bars = plt.barh(
    top_50_resultados['civilizacion'] + " vs " +
top_50_resultados['oponente'],
    top_50_resultados['winrate_percent'],
    color=[colores_civilizaciones[civ] for civ in
top_50_resultados['civilizacion']]
)

plt.ylabel('Civilización Ganadora vs Civilización Perdedora')
plt.xlabel('Win Rate (%)')
plt.title('Top 10 Win Rates (Tasas de Victoria) por Civilización
contra otra Civilización')
plt.xlim(0, 100)

for bar in bars:
    xval = bar.get_width()
    plt.text(xval + 1, bar.get_y() + bar.get_height() / 2,
f'{round(xval, 1)}%', va='center', fontsize=10)

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

```
<ipython-input-30-e307d252c369>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  top_50_resultados['winrate_percent'] =
(top_50_resultados['victorias'] /
```




```
resultados_finales_df['winrate_percent'] =
(resultados_finales_df['victorias'] /

(resultados_finales_df['victorias'] +
resultados_finales_df['derrotas'])) * 100

resultados_finales_df_ordenado_bajo =
resultados_finales_df.sort_values(by='winrate_percent',
ascending=True)
bottom_10_resultados = resultados_finales_df_ordenado_bajo.head(10)

plt.figure(figsize=(12, 10))
bars = plt.barh(
    bottom_10_resultados['civilizacion'] + " vs " +
bottom_10_resultados['oponente'],
    bottom_10_resultados['winrate_percent'],
    color=[colores_civilizaciones.get(civ, 'red') for civ in
bottom_10_resultados['civilizacion']]
```

```
)

plt.ylabel('Civilización Ganadora vs Civilización Perdedora')
plt.xlabel('Win Rate (%)')
plt.title('Bottom 10 Win Rates (Tasas de Victoria más Bajas) por
Civilización contra otra Civilización')
plt.xlim(0, 100)

for bar in bars:
    xval = bar.get_width()
    plt.text(xval + 1, bar.get_y() + bar.get_height() / 2,
f'{round(xval, 1)}%', va='center', fontsize=10)

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



Como podemos ver en el gráfico de barras hay nueve (9) civilizaciones con un winrate mayor o igual a 60%, por lo cual tendría posiblemente una mayor ventaja a la otra civilización solo por enfrentarse a esa, ocupando el ejemplo de los Franks vs Koreans, los Franks tienen un 23.8%

más de probabilidades de ganar que los Koreans, esto solo por ser los Franks y Koreans, ya que podemos asumir que los Franks son el counter de los Koreans, esto puede ser por múltiples factores como las tropas que pueden hacer un mayor daño o tienen una ventaja contra éstos, por lo que considerarems la hipótesis como correcta, hay una o más civilizaciones que son counter de otras.

As we can see in the bar graph, there are nine (9) civilizations with a win rate greater than or equal to 60%, which would possibly give them a greater advantage over the other civilization just by facing them. Taking the example of the Franks vs Koreans, the Franks have a 23.8% higher chance of winning than the Koreans, simply because they are the Franks and Koreans, as we can assume that the Franks counter the Koreans. This could be due to multiple factors such as troops that can deal more damage or have an advantage against them, so we will consider the hypothesis correct: there is one or more civilizations that counter others.

¿En los distintos rangos cambian los winrates de las civilizaciones?

Para el primer bloque de los dos que se ocuparán (Sin contar los otros dos que serán para graficar los resultados) se realizará un filtro para obtener los resultados entre los 500 y 1500 ratings. Se hace el conteo del número de victorias y derrotas para cada combinación, se renombran las columnas y se combinan los datos, se hace el calculo del winrate y se renombran nuevamente las columnas, finalmente se ordena la columna winrate en porcentaje mayor a menor.

For the first of the two blocks that will be used (not counting the other two that will be for graphing the results), a filter will be applied to obtain the results between 500 and 1500 ratings. The number of wins and losses for each combination is counted, the columns are renamed and the data is combined, the winrate is calculated and the columns are renamed again, and finally, the winrate column is sorted from highest to lowest percentage.

```
filtro_rating = particion_final[
    (particion_final['promedio_rating'] >= 500) &
    (particion_final['promedio_rating'] <= 1500)
]

victorias = filtro_rating.groupby(['civilizacion_win',
'civilizacion_lose']).size().reset_index(name='victorias')
derrotas = filtro_rating.groupby(['civilizacion_lose',
'civilizacion_win']).size().reset_index(name='derrotas')

victorias.rename(columns={'civilizacion_win': 'civilizacion',
'civilizacion_lose': 'oponente'}, inplace=True)
derrotas.rename(columns={'civilizacion_lose': 'civilizacion',
'civilizacion_win': 'oponente'}, inplace=True)

winrate_df_1500 = pd.merge(victorias, derrotas, on=['civilizacion',
'oponente'], how='outer').fillna(0)
winrate_df_1500['total_juegos'] = winrate_df_1500['victorias'] +
winrate_df_1500['derrotas']
```

```
winrate_df_1500['winrate'] = winrate_df_1500['victorias'] /
winrate_df_1500['total_juegos']
winrate_df_1500['winrate_percent'] = winrate_df_1500['winrate'] * 100
winrate_df_1500.columns = ['Civilización', 'Oponente', 'Victorias',
'Derrotas', 'Total Juegos', 'Win Rate', 'Win Rate (%)']
winrate_df_1500 = winrate_df_1500.sort_values(by='Win Rate (%)',
ascending=False)
```

Se realiza el mismo procedimiento pero para las partidas que se encuentren en un rating de entre 1500 y 2500.

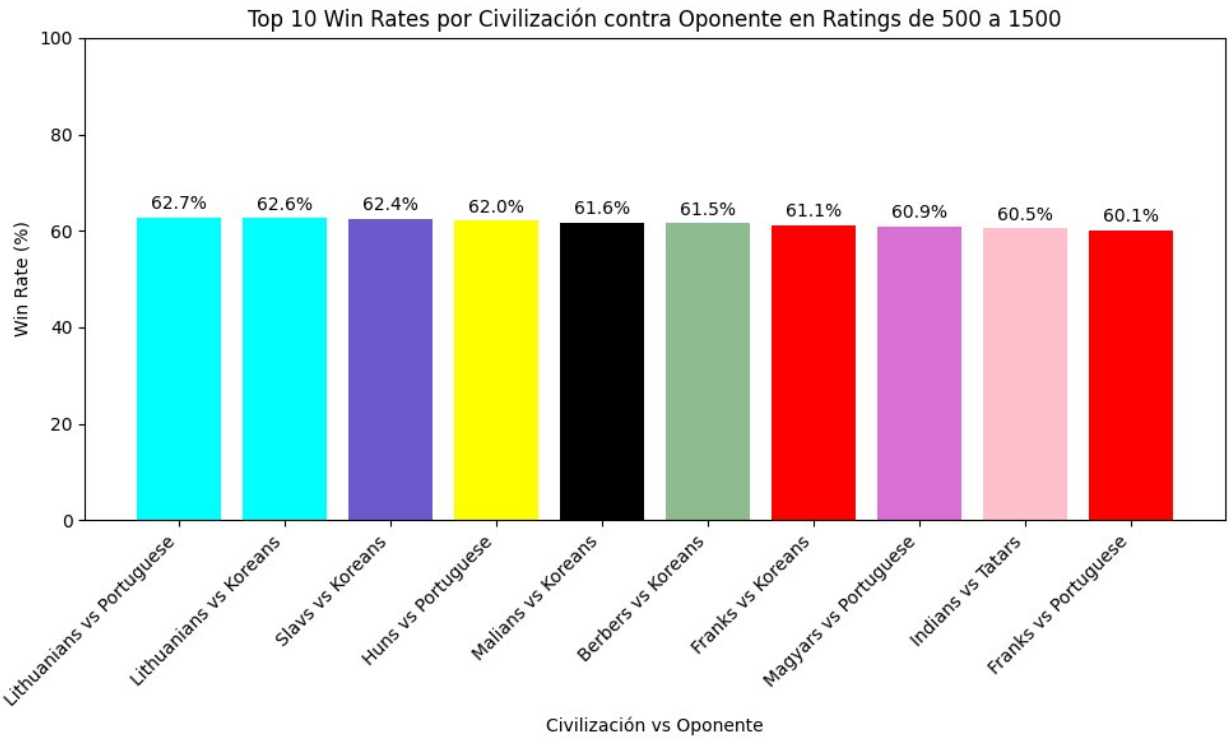
```
filtro_rating = particion_final[
    (particion_final['promedio_rating'] >= 1500) &
    (particion_final['promedio_rating'] <= 2500)
]

victorias = filtro_rating.groupby(['civilizacion_win',
'civilizacion_lose']).size().reset_index(name='victorias')
derrotas = filtro_rating.groupby(['civilizacion_lose',
'civilizacion_win']).size().reset_index(name='derrotas')

victorias.rename(columns={'civilizacion_win': 'civilizacion',
'civilizacion_lose': 'oponente'}, inplace=True)
derrotas.rename(columns={'civilizacion_lose': 'civilizacion',
'civilizacion_win': 'oponente'}, inplace=True)

winrate_df_2500 = pd.merge(victorias, derrotas, on=['civilizacion',
'oponente'], how='outer').fillna(0)
winrate_df_2500['total_juegos'] = winrate_df_2500['victorias'] +
winrate_df_2500['derrotas']
winrate_df_2500['winrate'] = winrate_df_2500['victorias'] /
winrate_df_2500['total_juegos']
winrate_df_2500['winrate_percent'] = winrate_df_2500['winrate'] * 100
winrate_df_2500.columns = ['Civilización', 'Oponente', 'Victorias',
'Derrotas', 'Total Juegos', 'Win Rate', 'Win Rate (%)']
winrate_df_2500 = winrate_df_2500.sort_values(by='Win Rate (%)',
ascending=False)
```

Se crea el gráfico para las partidas de entre 500 a 1500 de rating, mostrando las 10 civilizaciones con un mejor winrate.

The chart is created for matches with a rating between 500 and 1500, showing the 10 civilizations with the best win rate.

```
top_10_resultados = winrate_df_1500.head(10)
bar_colors = [colores_civilizaciones[civ] for civ in
top_10_resultados['Civilización']]

plt.figure(figsize=(10, 6))
```

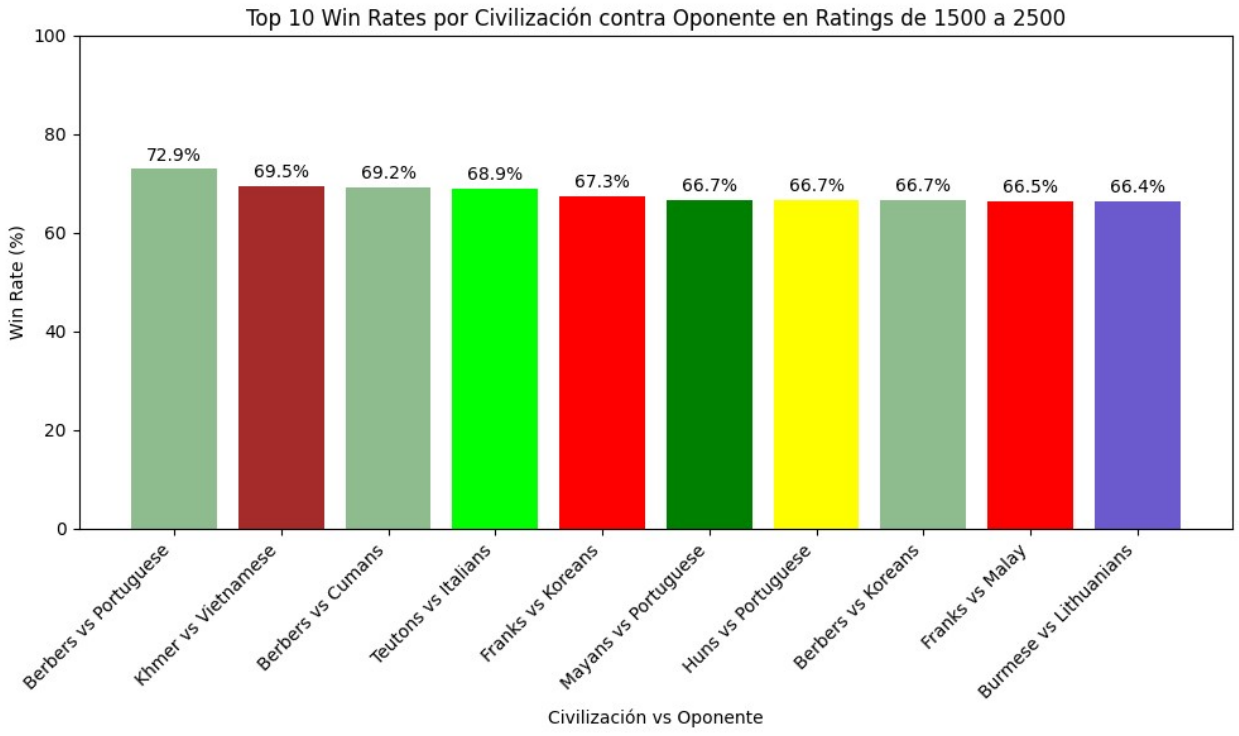
```
bars = plt.bar(
    top_10_resultados['Civilización'] + " vs " +
top_10_resultados['Oponente'],
    top_10_resultados['Win Rate (%)'],
    color=bar_colors
)

plt.xticks(rotation=45, ha='right')
plt.xlabel('Civilización vs Oponente')
plt.ylabel('Win Rate (%)')
plt.title('Top 10 Win Rates por Civilización contra Oponente en
Ratings de 500 a 1500')
plt.ylim(0, 100)

for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width() / 2, yval + 1,
f'{round(yval, 1)}%', ha='center', va='bottom', fontsize=10)

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



Se crea el gráfico para las partidas de entre 1500 a 2500 de rating, mostrando las 10 civilizaciones con un mejor winrate.

The graph is created for matches with a rating between 1500 and 2500, showing the 10 civilizations with the best win rate.

```
top_10_resultados = winrate_df_2500.head(10)
bar_colors = [colores_civilizaciones[civ] for civ in
top_10_resultados['Civilización']]

plt.figure(figsize=(10, 6))
bars = plt.bar(
    top_10_resultados['Civilización'] + " vs " +
top_10_resultados['Oponente'],
    top_10_resultados['Win Rate (%)'],
    color=bar_colors
)

plt.xticks(rotation=45, ha='right')
plt.xlabel('Civilización vs Oponente')
plt.ylabel('Win Rate (%)')
plt.title('Top 10 Win Rates por Civilización contra Oponente en
Ratings de 1500 a 2500')
plt.ylim(0, 100)

for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width() / 2, yval + 1,
f'{round(yval, 1)}%', ha='center', va='bottom', fontsize=10)

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



**Como podemos ver en los gráficos de barras entre las diez (10) civilizaciones con su respectivo winrate, se repiten 2 enfrentamientos entre los rangos, Franks vs Koreans que entre los rangos 500 a 1500 (rating) tienen un winrate de 61.4%, cuando entre los rangos 1500 a 2500 (rating) tienen un winrate del 71.2%, siendo mayor en los rangos más altos. Pasando al otro dato que se puede ver entre los graficos están los Goths vs Indians, donde entre los rangos 500 a 1500 tienen un winrate del 60.1% y entre los rangos 1500 a 2500 tienen un winrate del 66.7%.

Podemos asumir viendo el resto de datos que entre mayor sea el rango se encontrará una diferencia entre el winrate o una mayor ventaja de ciertas civilizaciones al tener un enfrentamiento entre otras, donde entre los Franks vs Koreans la diferencia del winrate es del 10.9%, por lo que tiene una mayor ventaja en los rangos superiores, por lo que podemos dar la hipótesis como correcta, hay al menos una civilización con un counter que tiene una diferencia mayor al 10%.**

Ahora se pasa a hacer la limpieza de datos para los modelos de regresión y clasificación (Creación del detaset para los modelos con las columnas nuevas)[Fase 3]

As we can see in the bar charts among the ten (10) civilizations with their respective win rates, there are 2 matchups that repeat across the ranks, Franks vs Koreans, which have a win rate of 61.4% between the 500 to 1500 (rating) range, while between the 1500 to 2500 (rating) range, they have a win rate of 71.2%, being higher in the higher ranks. Moving on to the other data that can be seen in the graphs, we have Goths vs Indians, where between the ranks of 500 to 1500 they have a win rate of 60.1% and between the ranks of 1500 to 2500 they have a win rate of 66.7%.

We can assume, looking at the rest of the data, that the higher the rank, the greater the difference in winrate or the greater the advantage of certain civilizations when facing others. For example, between the Franks and Koreans, the winrate difference is 10.9%, giving a greater advantage in the higher ranks. Therefore, we can consider the hypothesis correct: there is at least one civilization with a counter that has a difference greater than 10%.

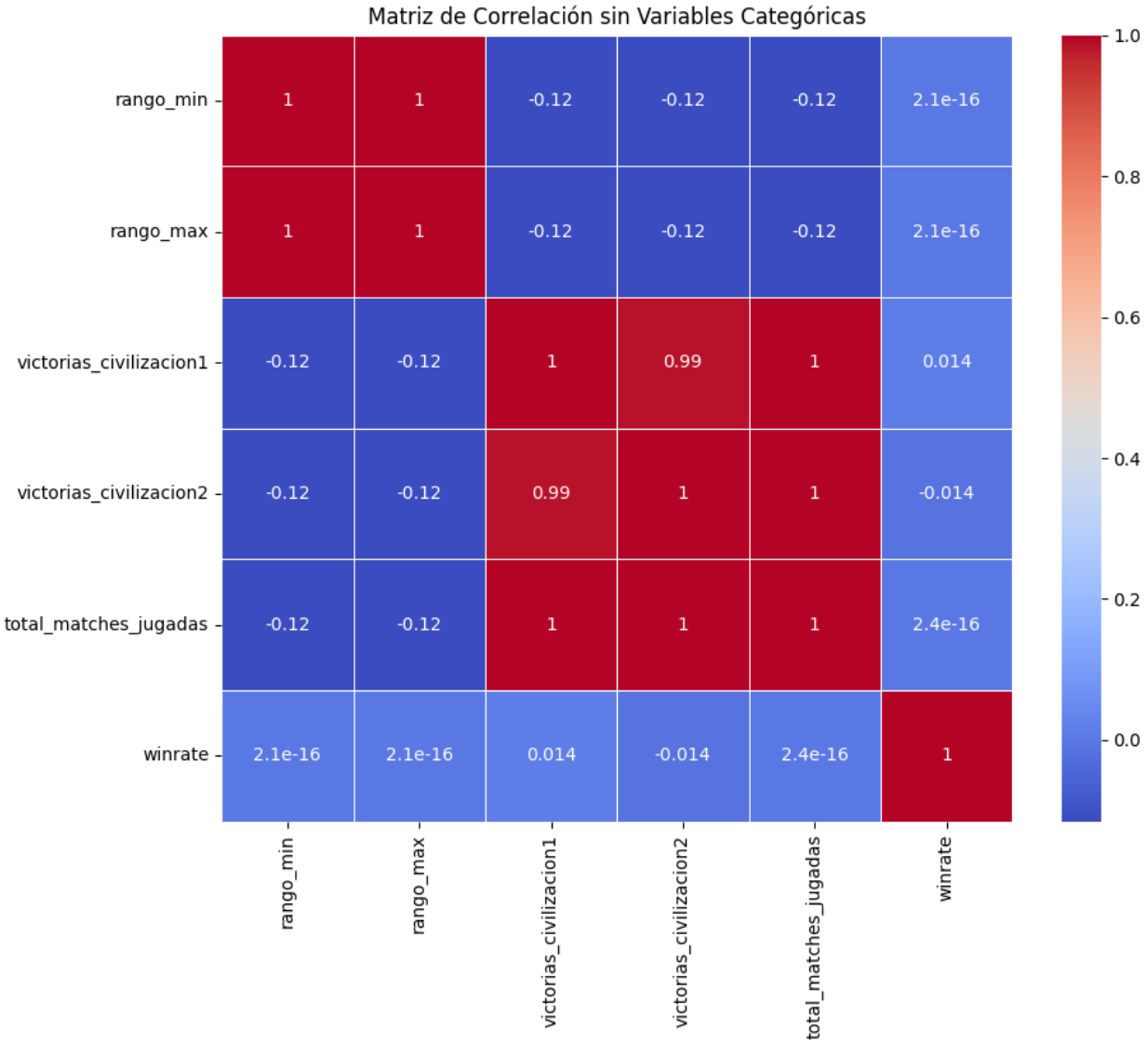
Now we move on to data cleaning for regression and classification models (Creation of the dataset for the models with the new columns) [Phase 3]

Matrices de correlación

Matriz de correlación para valores numéricos continuos o discretos.

Correlation matrix for continuous or discrete numerical values.

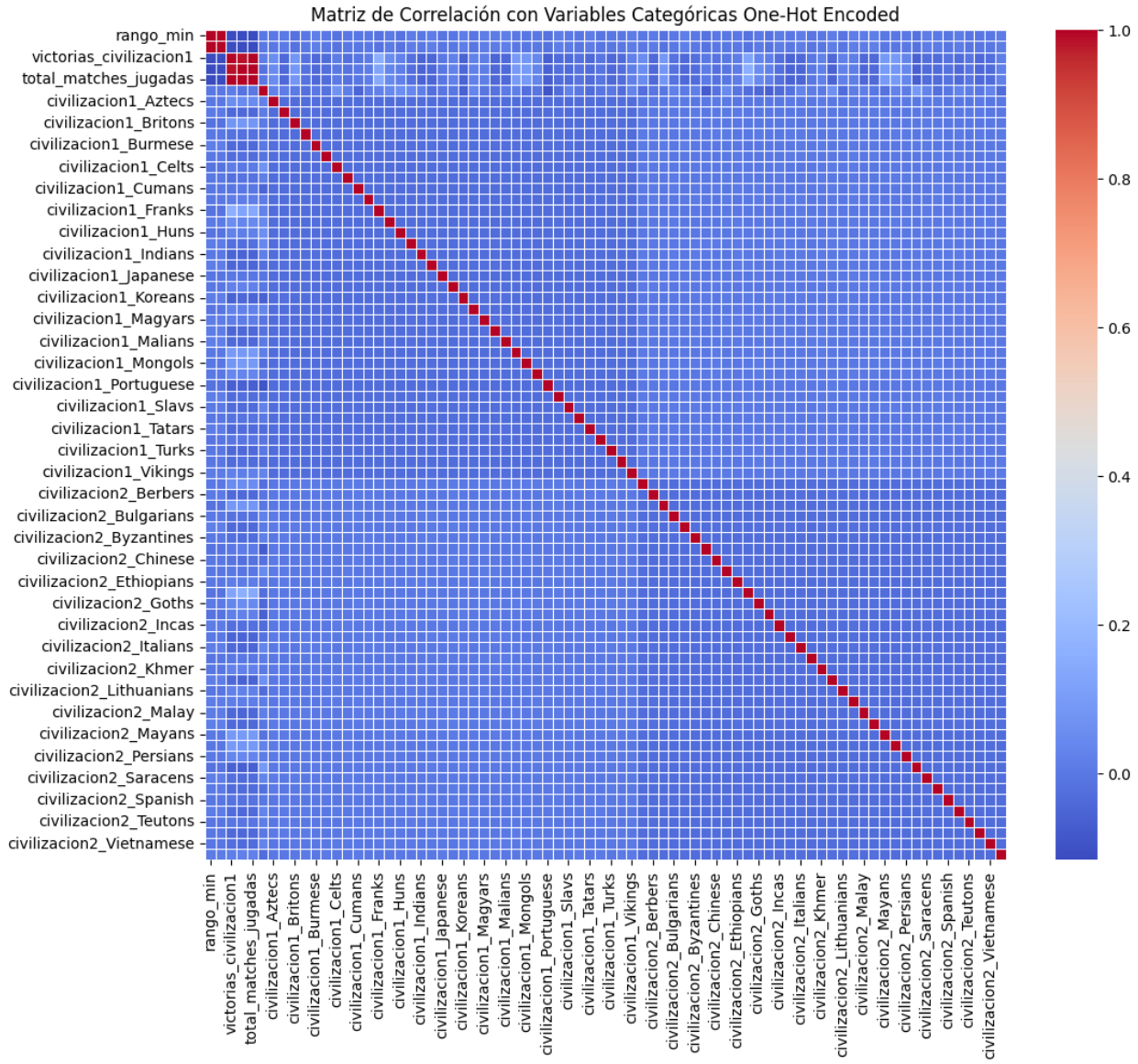
```
resultados_numericos = resultados.drop(columns=['civilizacion1',
'civilizacion2'])
correlation_matrix = resultados_numericos.corr()

plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm',
linewidths=0.5)
plt.title('Matriz de Correlación sin Variables Categóricas')
plt.show()
```



Matriz de correlación con los datos categóricos

Correlation matrix with categorical data

```
resultados_encoded = pd.get_dummies(resultados,
columns=['civilizacion1', 'civilizacion2'])
correlation_matrix_encoded = resultados_encoded.corr()

plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix_encoded, annot=False, cmap='coolwarm',
linewidths=0.5)
plt.title('Matriz de Correlación con Variables Categóricas One-Hot
Encoded')
plt.show()
```



Ahora se pasa a la creación de los modelos de clasificación y regresión. [Fase 4]

Now we move on to the creation of classification and regression models. [Phase 4]

```
# Insertar cuantos bloques de código consideren necesarios

# Se recomienda obtener estadísticos descriptivos para apoyar
hipótesis inferenciales.
# Reconocer la naturaleza de los datos y como tratarlos en etapas
posteriores y dar ideas de como se podría transformar.
# Identificar MissingValues, outliers, medidas de posición, medidas de
dispersión etc.
```

Fase 3: Data Preparation

RATING A ENTERO

Se realiza la conversión ya que se trabajará con el promedio de éste y se necesita que sea un entero

The conversion is performed since we will be working with its average and it needs to be an integer.

```
aoe['rating'] = pd.to_numeric(aoe['rating'], errors='coerce')
```

Luego de pasar a entero la columna rating se puede obtener el promedio [Fase 2]

After converting the rating column to integers, the average can be obtained [Phase 2]

LLenado de datos nulos

Función para sacar el promedio del rating por la partida y insertar el promedio de esa partida en las filas vacías. (Si el match es nulo no se puede rellenar ese dato) [Tarda aproximadamente 4min]

Function to calculate the average rating per game and insert the average of that game into the empty rows. (If the match is null, that data cannot be filled in) [Takes approximately 4 minutes]

```
def fill_nulos_rating(row):
    if pd.isnull(row['rating']):
        if pd.isnull(row['match']):
            return row['rating']
        else:
            return promedio_rating_match.get(row['match'],
row['rating'])
    else:
        return row['rating']

aoe['rating'] = aoe.apply(fill_nulos_rating, axis=1)
```

Ahora se debe pasar a la fase 2 para verificar si quedan valores nulos (VERIFICAR VALORES NULOS RATING) [Fase 2]

Now we must move to phase 2 to check if there are any null values (CHECK NULL VALUES RATING) [Phase 2]

ELIMINACIÓN DE LOS OTROS NAN

```
aoe_2 = aoe_2.dropna()
```

*Pasaremos una última verificación en la Fase 2 [Última verificación Nulos] [Fase 2]

*We will conduct a final check in Phase 2 [Final Null Check] [Phase 2]

TRANSFORMACIÓN DE WINNER

Transformaremos la columna en enteros [0 y 1], al igual que verificaremos que la conversión haya sido exitosa.

We will transform the column into integers [0 and 1], and we will also verify that the conversion was successful.

```
aoe_2_sin_outliers['winner'] =
aoe_2_sin_outliers['winner'].astype(int)

print(aoe_2_sin_outliers['winner'].unique())
```

```
[0 1]

<ipython-input-19-4141e50bcbc6>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  aoe_2_sin_outliers['winner'] =
aoe_2_sin_outliers['winner'].astype(int)
```

Con la transformación del Winner para más adelante pasaremos a la fase 2 para responder la primera pregunta, ¿Qué civilización es la más jugada? [Fase 2]

With the transformation of the Winner, we will move on to phase 2 to answer the first question, which civilization is the most played? [Phase 2]

Creación del detaset para los modelos con las columnas nuevas

Se crean las civilizaciones1 y la civilización2, donde irán las civilizaciones que se enfrentan entre los rangos en el rating, luego se hacen los rangos min y máximos de las civilizaciones que se encontrarán entre las filas min y max, luego se obtienen el conteo de victorias de cada civilización y se hace un merge renombrando las columnas en el dataframe, para finalizar se obtiene el calculo de las partidas totales y el winrate de la civilización1 contra la civilización2

Civilization1 and Civilization2 are created, where the civilizations that face each other within the rating ranges will go. Then, the minimum and maximum ranks of the civilizations that will be found between the min and max rows are established. Next, the victory count of each civilization is obtained, and a merge is performed, renaming the columns in the dataframe. Finally, the total number of matches and the win rate of Civilization1 against Civilization2 are calculated.

```
enfrentamientos = pd.DataFrame()
enfrentamientos['civilizacion1'] =
modelo_regresion_datos1[['civilizacion_win',
'civilizacion_lose']].min(axis=1)
enfrentamientos['civilizacion2'] =
modelo_regresion_datos1[['civilizacion_win',
'civilizacion_lose']].max(axis=1)


modelo_regresion_datos1['rango_min'] =
(modelo_regresion_datos1['promedio_rating'] // 400) * 400
modelo_regresion_datos1['rango_max'] =
modelo_regresion_datos1['rango_min'] + 400

victorias_civilizacion1 =
modelo_regresion_datos1.groupby(['rango_min', 'rango_max',
'civilizacion_win',
'civilizacion_lose']).size().reset_index(name='victorias_civilizacion1
')
victorias_civilizacion2 =
modelo_regresion_datos1.groupby(['rango_min', 'rango_max',
'civilizacion_lose',
'civilizacion_win']).size().reset_index(name='victorias_civilizacion2'
)

victorias_civilizacion2.rename(columns={'civilizacion_lose':
'civilizacion1', 'civilizacion_win': 'civilizacion2'}, inplace=True)
victorias_civilizacion1.rename(columns={'civilizacion_win':
'civilizacion1', 'civilizacion_lose': 'civilizacion2'}, inplace=True)
resultados = pd.merge(victorias_civilizacion1,
victorias_civilizacion2, on=['civilizacion1', 'civilizacion2',
'rango_min', 'rango_max'], how='outer').fillna(0)

resultados['total_matches_jugadas'] =
resultados['victorias_civilizacion1'] +
resultados['victorias_civilizacion2']
resultados['winrate'] = resultados['victorias_civilizacion1'] /
resultados['total_matches_jugadas']

resultados
```


```
{"summary":"{\n  \"name\": \"resultados\",\n  \"rows\": 6790,\n
\"fields\": [\n    {\n      \"column\": \"rango_min\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
644.9338508466145,\n        \"min\": 0.0,\n        \"max\": 2000.0,\n
\"num_unique_values\": 6,\n        \"samples\": [\n          400.0,\n
800.0,\n          0.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"rango_max\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 644.9338508466145,\n        \"min\":
```

```
400.0,\n        \"max\": 2400.0,\n        \"num_unique_values\": 6,\n
\"samples\": [\n          800.0,\n          1200.0,\n          400.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"civilizacion1\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 35,\n        \"samples\": [\n
\"Portuguese\",\n          \"Incas\",\n          \"Mongols\"\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"civilizacion2\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 35,\n
\"samples\": [\n          \"Portuguese\",\n          \"Incas\",\n
\"Mongols\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"victorias_civilizacion1\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 579.1918805777203,\n
\"min\": 0.0,\n        \"max\": 10463.0,\n
\"num_unique_values\": 1345,\n        \"samples\": [\n
860.0,\n          529.0,\n          1492.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"victorias_civilizacion2\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
579.1918805777201,\n        \"min\": 0.0,\n        \"max\": 10463.0,\n
\"num_unique_values\": 1345,\n        \"samples\": [\n
735.0,\n          1767.0,\n          1709.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"total_matches_jugadas\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1154.5181042961885,\n        \"min\": 1.0,\n        \"max\": 20926.0,\
n        \"num_unique_values\": 1333,\n        \"samples\": [\n
12310.0,\n          230.0,\n          1437.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"winrate\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
0.14490323270900762,\n        \"min\": 0.0,\n        \"max\": 1.0,\n
\"num_unique_values\": 3895,\n        \"samples\": [\n
0.45394736842105265,\n          0.5518394648829431,\n
0.45121951219512196\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"resultados"}
```


Ahora se pasa a hacer las matrices de correlación para obtener las relaciones de las columnas (Matrices de correlación)[Fase 2]

Now we move on to creating the correlation matrices to obtain the relationships of the columns (Correlation Matrices) [Phase 2]

```
# Insertar cuantos bloques de código consideren necesarios
```

```
# Se recomienda considerar todas las transformaciones necesarias para
obtener la data lo más limpia posible.
# Realizar tratamiento a todos los datos que consideren necesarios.
```

Fase 4: Modeling

Regresión

Con el modelo de regresión se tiene como objetivo el predecir el winrate de una civilización en un rango estimado, este se define como el porcentaje de partidas ganadas en comparación con el total de partidas jugadas. Esta métrica es fundamental para evaluar el desempeño y la efectividad de una civilización en el juego y busca obtener una mayor seguridad a momento de enfrentar 2 civilizaciones.

Selección de variables independientes y dependiente

With the regression model, the objective is to predict the winrate of a civilization within an estimated range, defined as the percentage of games won compared to the total games played. This metric is fundamental for evaluating the performance and effectiveness of a civilization in the game and aims to achieve greater security when facing two civilizations.

Selection of independent and dependent variables

```
X = resultados[['rango_min', 'rango_max', 'victorias_civilizacion1',
'victorias_civilizacion2']]
y = resultados['winrate']
```

Se dividen los datos en 20% de prueba y 80% para entrenar los modelos

The data is divided into 20% for testing and 80% for training the models.

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

Modelo de regresión KNeighborsRegressor con ElasticNetCV

Se crea y entrena el modelo ElasticNetCV

The ElasticNetCV model is created and trained.

```
elastic_net = ElasticNetCV(l1_ratio=0.6, tol=0.1, random_state=42)
elastic_net.fit(X_train, y_train)
```

```
ElasticNetCV(l1_ratio=0.6, random_state=42, tol=0.1)
```

Se usan las predicciones del modelo ElasticNetCV como entrada para el modelo KNeighborsRegressor

The predictions from the ElasticNetCV model are used as input for the KNeighborsRegressor model.

```
X_train_elastic_net = elastic_net.predict(X_train).reshape(-1, 1)
X_test_elastic_net = elastic_net.predict(X_test).reshape(-1, 1)
```

Se crea el modelo KNeighborsRegressor con 2 vecinos, se entrena el modelo y se realiza la predicción

The KNeighborsRegressor model is created with 2 neighbors, the model is trained, and the prediction is made.

```
knn = KNeighborsRegressor(n_neighbors=2, p=2, weights='distance')
knn.fit(X_train_elastic_net, y_train)
y_pred_knn = knn.predict(X_test_elastic_net)
```

Se calculan las métricas del modelo y se muestran los resultados de estas

The model metrics are calculated and the results are displayed.

```
mae_KNeighborsRegressor = mean_absolute_error(y_test, y_pred_knn)
mse_KNeighborsRegressor = mean_squared_error(y_test, y_pred_knn)
rmse_KNeighborsRegressor = np.sqrt(mse_KNeighborsRegressor)
r2_KNeighborsRegressor = r2_score(y_test, y_pred_knn)

print("Mean Absolute Error (MAE):", mae_KNeighborsRegressor)
print("Mean Squared Error (MSE):", mse_KNeighborsRegressor)
print("Root Mean Squared Error (RMSE):", rmse_KNeighborsRegressor)
print("Score del modelo (R²):", r2_KNeighborsRegressor)
```

```
Mean Absolute Error (MAE): 0.023969706582109968
Mean Squared Error (MSE): 0.0020063518119173736
Root Mean Squared Error (RMSE): 0.0447923186709214
Score del modelo (R²): 0.9082273827382688
```

Se obtienen los valores residuales

The residual values are obtained.

```
residuals = y_test - y_pred_knn
```

Creación del gráfico de comparación de valores reales vs las predicciones

Creation of the chart comparing actual values vs predictions

```
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
scatter = plt.scatter(y_test, y_pred_knn, c=np.abs(residuals),
cmap='coolwarm', alpha=0.8)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], '--',
```

```
color='red', label="Línea ideal")
plt.colorbar(scatter, label='Magnitud del error')
plt.xlabel('Valores reales')
plt.ylabel('Predicciones')
plt.title('Valores Reales vs. Predicciones (Coloreado por error)')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x79e91cf6a1d0>
```

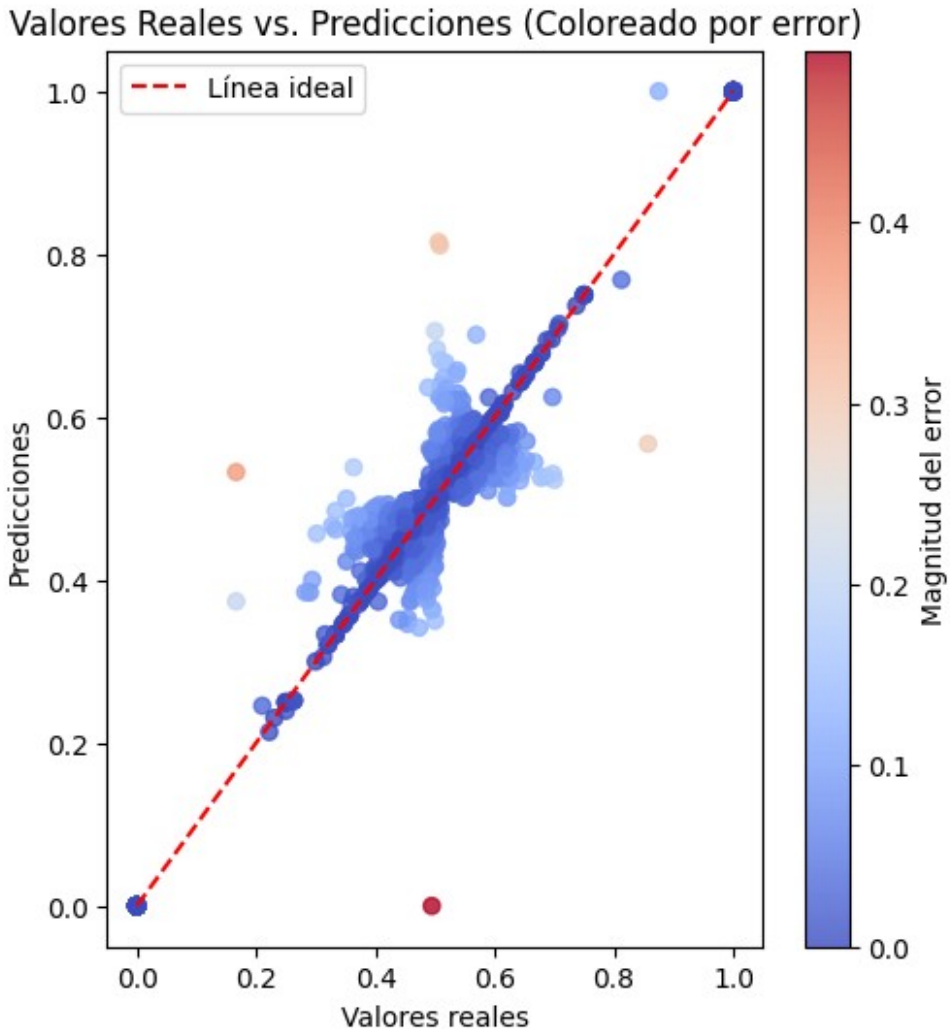


Gráfico de distribución de errores con historigrama

Error distribution chart with histogram

```
plt.subplot(1, 2, 2)
sns.histplot(residuals, bins=20, kde=True, color='blue',
stat="density", label="Histograma")
plt.axvline(0, color='red', linestyle='--', label='Error 0')
plt.xlabel('Errores')
```

```
plt.ylabel('Densidad')
plt.title('Distribución de Errores (con KDE)')
plt.legend()
plt.tight_layout()
plt.show()
```

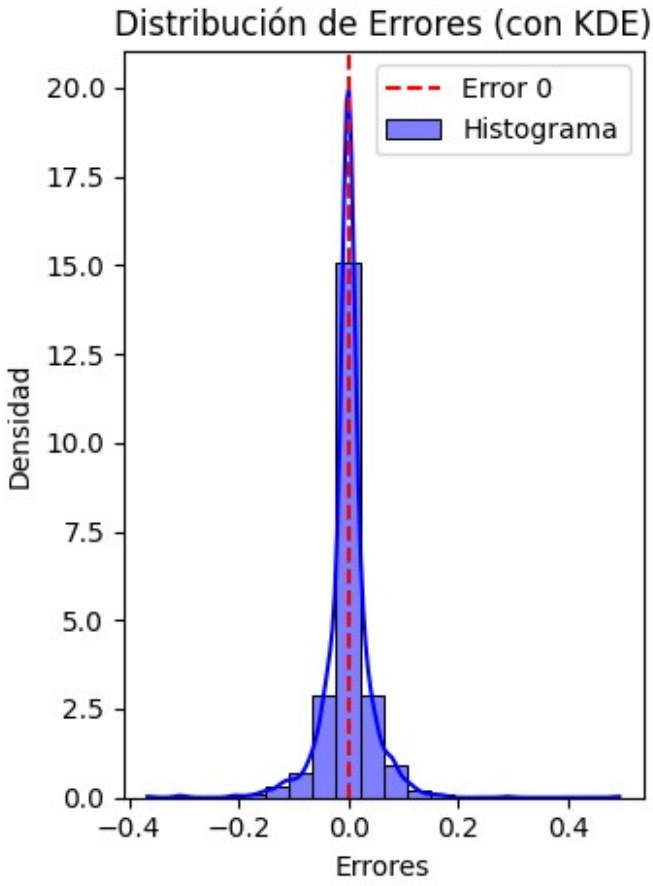
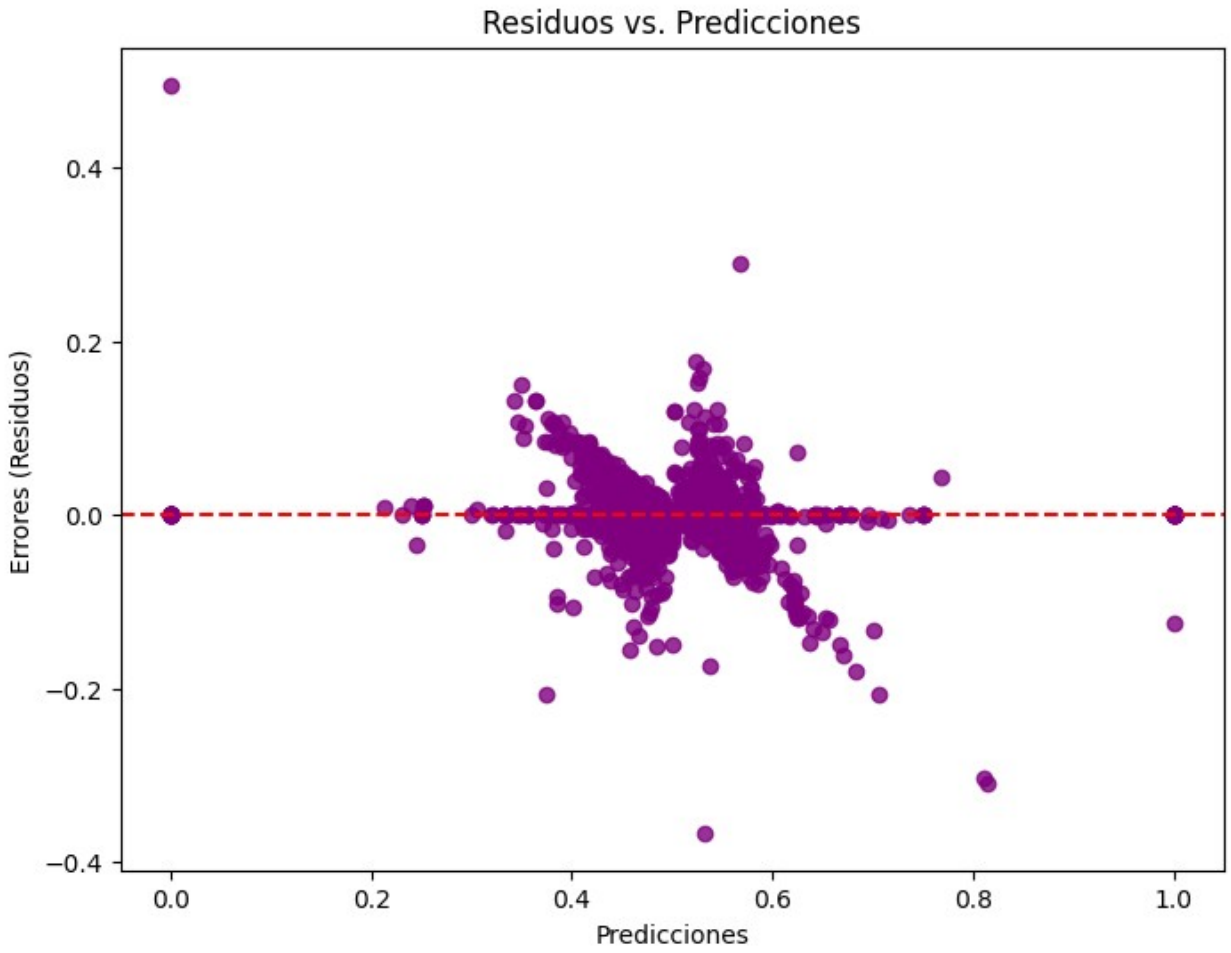


Gráfico de valores residuales vs las predicciones

Residuals vs. Predicted Values Plot

```
plt.figure(figsize=(8, 6))
plt.scatter(y_pred_knn, residuals, alpha=0.8, color='purple')
plt.axhline(0, color='red', linestyle='--')
plt.xlabel('Predicciones')
plt.ylabel('Errores (Residuos)')
plt.title('Residuos vs. Predicciones')
plt.show()
```



**Modelo de regresión Random Forest Regressor*

Se crea, se entrena y se realizan las predicciones del modelo RandomForestRegressor

The RandomForestRegressor model is created, trained, and predictions are made.

```
random_forest = RandomForestRegressor(n_estimators=100,
random_state=42)
random_forest.fit(X_train, y_train)
y_pred_rf = random_forest.predict(X_test)
```

Se realizan las métricas de rendimiento y se muestran los resultados

Performance metrics are conducted and the results are shown.

```
mae_RandomForestRegressor = mean_absolute_error(y_test, y_pred_rf)
mse_RandomForestRegressor = mean_squared_error(y_test, y_pred_rf)
rmse_RandomForestRegressor = np.sqrt(mse_RandomForestRegressor)
r2_RandomForestRegressor = r2_score(y_test, y_pred_rf)
```

```
print("Mean Absolute Error (MAE):", mae_RandomForestRegressor)
print("Mean Squared Error (MSE):", mse_RandomForestRegressor)
print("Root Mean Squared Error (RMSE):", rmse_RandomForestRegressor)
print("Score del modelo (R²):", r2_RandomForestRegressor)
```

```
Mean Absolute Error (MAE): 0.002691546733640345
Mean Squared Error (MSE): 3.0972048547968354e-05
Root Mean Squared Error (RMSE): 0.005565253682265379
Score del modelo (R²): 0.9985833063073379
```

Se calculan los errores del modelo

The model errors are calculated.

```
residuals_rf = y_test - y_pred_rf
```

Creación del gráfico de comparación de valores reales vs las predicciones

Creation of the chart comparing actual values vs predictions

```
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
scatter_rf = plt.scatter(y_test, y_pred_rf, c=np.abs(residuals_rf),
cmap='coolwarm', alpha=0.8)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], '--',
color='red', label="Línea ideal")
plt.colorbar(scatter_rf, label='Magnitud del error')
plt.xlabel('Valores reales')
plt.ylabel('Predicciones')
plt.title('Valores Reales vs. Predicciones (Coloreado por error)')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x79e8c5112fe0>
```

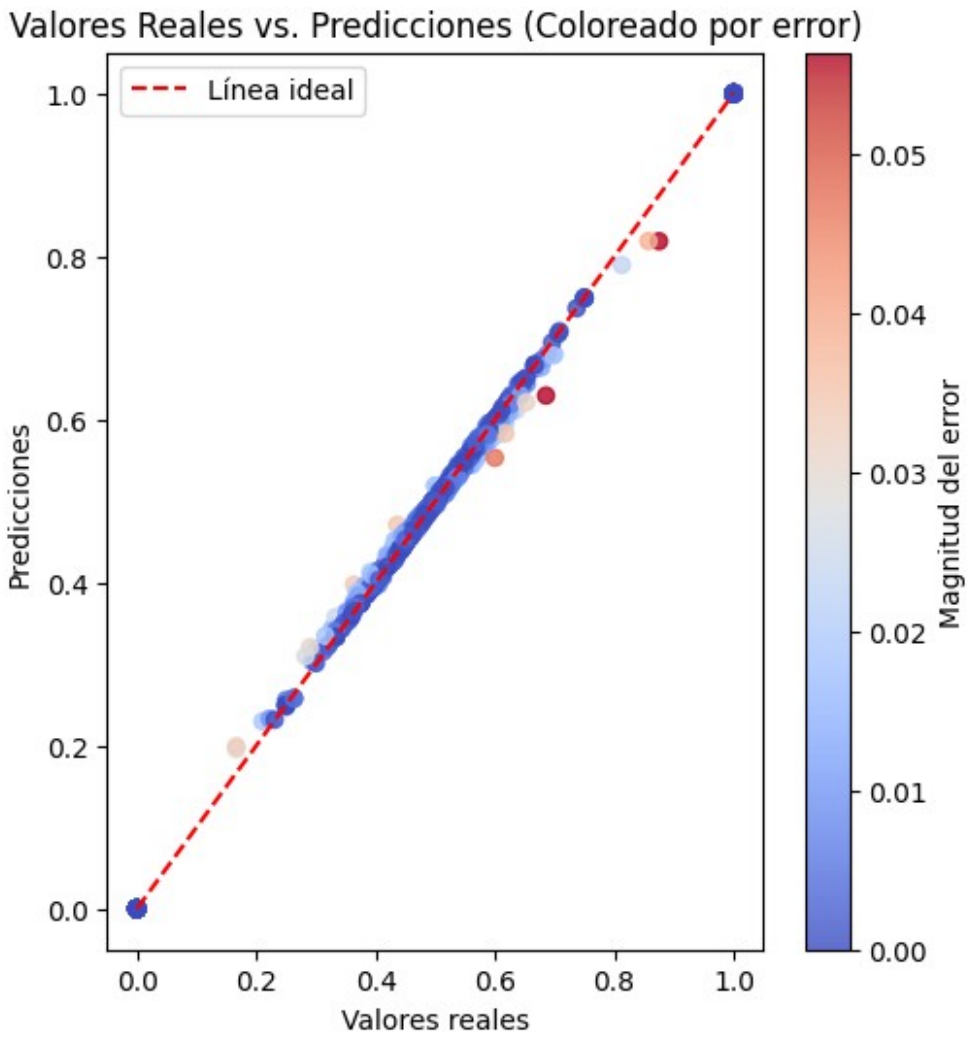


Gráfico de distribución de errores con historigrama

Error distribution chart with histogram

```
plt.subplot(1, 2, 2)
sns.histplot(residuals_rf, bins=20, kde=True, color='blue',
stat="density", label="Histograma")
plt.axvline(0, color='red', linestyle='--', label='Error 0')
plt.xlabel('Errores')
plt.ylabel('Densidad')
plt.title('Distribución de Errores (con KDE)')
plt.legend()
plt.tight_layout()
plt.show()
```

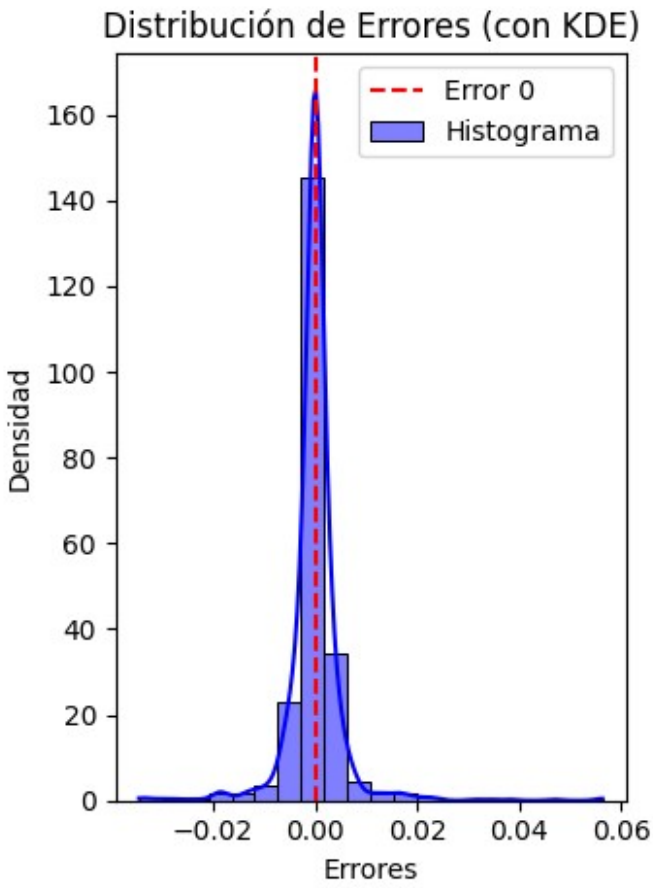
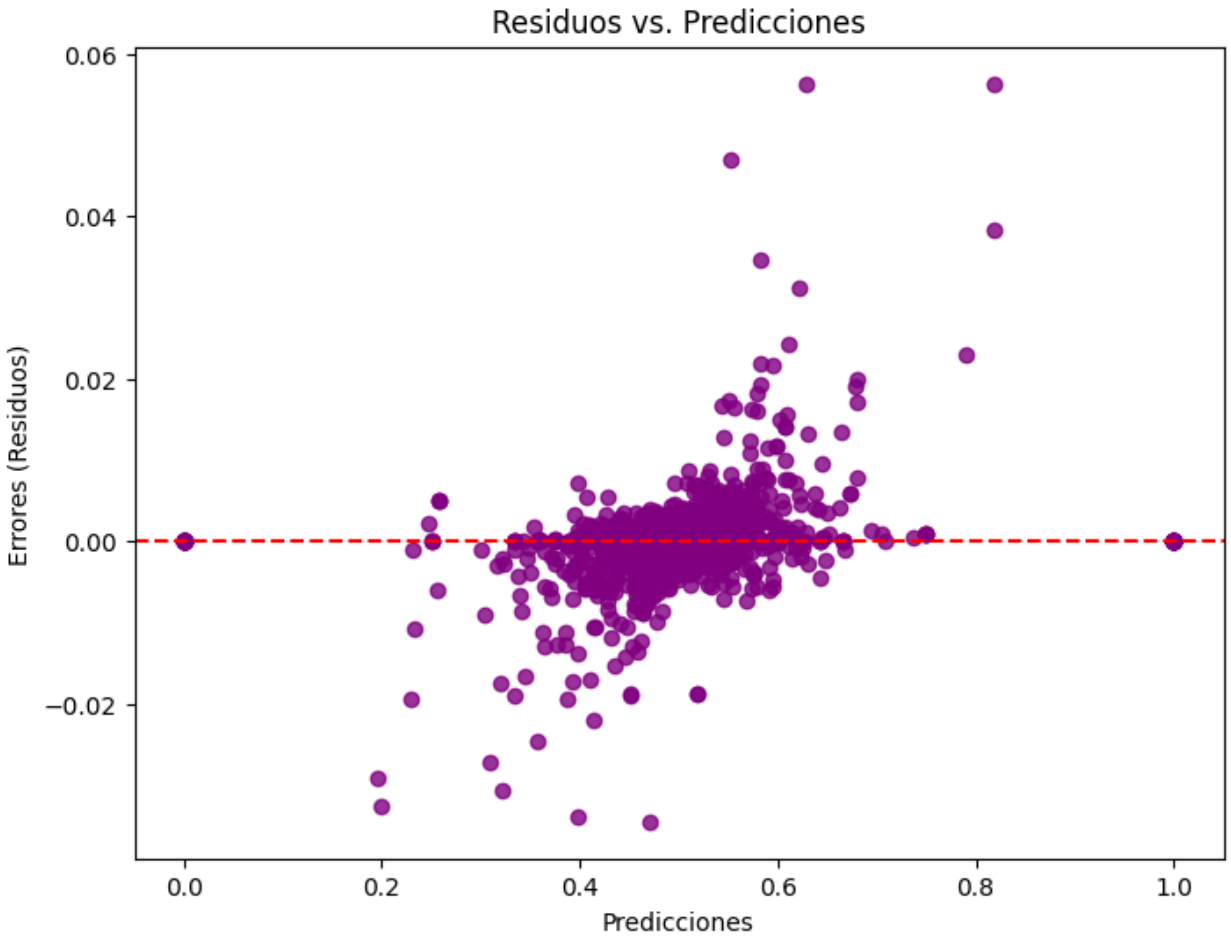


Gráfico de valores residuales vs las predicciones

Residuals vs. Predicted Values Plot

```
plt.figure(figsize=(8, 6))
plt.scatter(y_pred_rf, residuals_rf, alpha=0.8, color='purple')
plt.axhline(0, color='red', linestyle='--')
plt.xlabel('Predicciones')
plt.ylabel('Errores (Residuos)')
plt.title('Residuos vs. Predicciones')
plt.show()
```




```
# Insertar cuantos bloques de código consideren necesarios

# Realizar tarea de regresión de datos orientado al caso entregado
```

Clasificación

El modelo de clasificación tiene como objetivo predecir el margen de victoria entre dos civilizaciones en el juego Age of Empires II. Se busca identificar si una civilización tiene una ventaja significativa sobre otra en función de un umbral predefinido de diferencia en el número de victorias.

Se crea la variable que contenga el umbral de diferencia para saber si el margen de victoria que tiene una civilización es alta o baja (1 y 0 respectivamente), al igual que se crea la columna para el dataset

The classification model aims to predict the victory margin between two civilizations in the game Age of Empires II. The goal is to identify if one civilization has a significant advantage over another based on a predefined threshold of difference in the number of victories.

The variable containing the threshold difference to determine if a civilization's victory margin is high or low (1 and 0 respectively) is created, as well as the column for the dataset.

```
umbral = 0.2
resultados['margen_victoria'] = resultados.apply(lambda row: 1 if
row['victorias_civilizacion1'] > row['victorias_civilizacion2'] * (1 +
umbral) else 0, axis=1)
```

Se selecciona las características independientes y dependientes del modelo de clasificación

The independent and dependent features of the classification model are selected.

```
X = resultados[['rango_min', 'rango_max', 'victorias_civilizacion1',
'victorias_civilizacion2', 'total_matches_jugadas']]
y = resultados['margen_victoria']
```

Se dividen los datos en 30% para la prueba y 70% para entrenar los modelos

The data is divided into 30% for testing and 70% for training the models.

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)
```

LogisticRegression

Realizar una búsqueda de hiperparámetros usando GridSearchCV y encontrar cual es el mejor valor para el parámetro, almacenandolo en una variable para ocuparlo en el modelo

Perform a hyperparameter search using GridSearchCV and find the best value for the parameter, storing it in a variable to use it in the model.

```
param_grid = {'C': [0.1, 1, 10]}
grid_search = GridSearchCV(LogisticRegression(solver='liblinear',
random_state=42), param_grid, cv=3)
grid_search.fit(X_train, y_train)
best_C = grid_search.best_params_['C']
print("Mejores parámetros:", grid_search.best_params_)
```

```
Mejores parámetros: {'C': 1}
```

Se crea, se entrena y se realizan las predicciones del modelo LogisticRegression

The LogisticRegression model is created, trained, and predictions are made.

```
logistic_model = LogisticRegression(C=best_C, dual=False,
penalty='l2', random_state=42)
logistic_model.fit(X_train, y_train)
y_pred_prob = logistic_model.predict_proba(X_test)[:, 1]
y_pred = (y_pred_prob >= umbral).astype(int)
y_pred_train = logistic_model.predict(X_train)
y_pred_prob_train = logistic_model.predict_proba(X_train)[:, 1]
```

Se calculan y muestran las métricas de rendimiento para el modelo LogisticRegression y las métricas del conjunto de pruebas para identificar si existe overfitting

The performance metrics for the LogisticRegression model and the metrics for the test set are calculated and displayed to identify if there is overfitting.

```
accuracy_LogisticRegression = accuracy_score(y_test, y_pred)
precision_LogisticRegression = precision_score(y_test, y_pred)
recall_LogisticRegression = recall_score(y_test, y_pred)
f1_LogisticRegression = f1_score(y_test, y_pred)
roc_auc_LogisticRegression = roc_auc_score(y_test, y_pred_prob)

accuracy_train_LogisticRegression = accuracy_score(y_train,
y_pred_train)
precision_train_LogisticRegression = precision_score(y_train,
y_pred_train)
recall_train_LogisticRegression = recall_score(y_train, y_pred_train)
f1_train_LogisticRegression = f1_score(y_train, y_pred_train)
roc_auc_train_LogisticRegression = roc_auc_score(y_train,
y_pred_prob_train)

print("Métricas en el conjunto de entrenamiento:")
print(f"Accuracy: {accuracy_train_LogisticRegression:.4f}")
print(f"Precision: {precision_train_LogisticRegression:.4f}")
print(f"Recall: {recall_train_LogisticRegression:.4f}")
print(f"F1-Score: {f1_train_LogisticRegression:.4f}")
print(f"AUC-ROC: {roc_auc_train_LogisticRegression:.4f}")

print("\nMétricas en el conjunto de prueba:")
print(f"Accuracy: {accuracy_LogisticRegression:.4f}")
print(f"Precision: {precision_LogisticRegression:.4f}")
print(f"Recall: {recall_LogisticRegression:.4f}")
print(f"F1-Score: {f1_LogisticRegression:.4f}")
print(f"AUC-ROC: {roc_auc_LogisticRegression:.4f}")
```

```
Métricas en el conjunto de entrenamiento:
Accuracy: 1.0000
Precision: 1.0000
Recall: 1.0000
F1-Score: 1.0000
AUC-ROC: 1.0000

Métricas en el conjunto de prueba:
Accuracy: 0.9946
Precision: 0.9763
Recall: 1.0000
F1-Score: 0.9880
AUC-ROC: 1.0000
```

Reporte de clasificación

Classification report

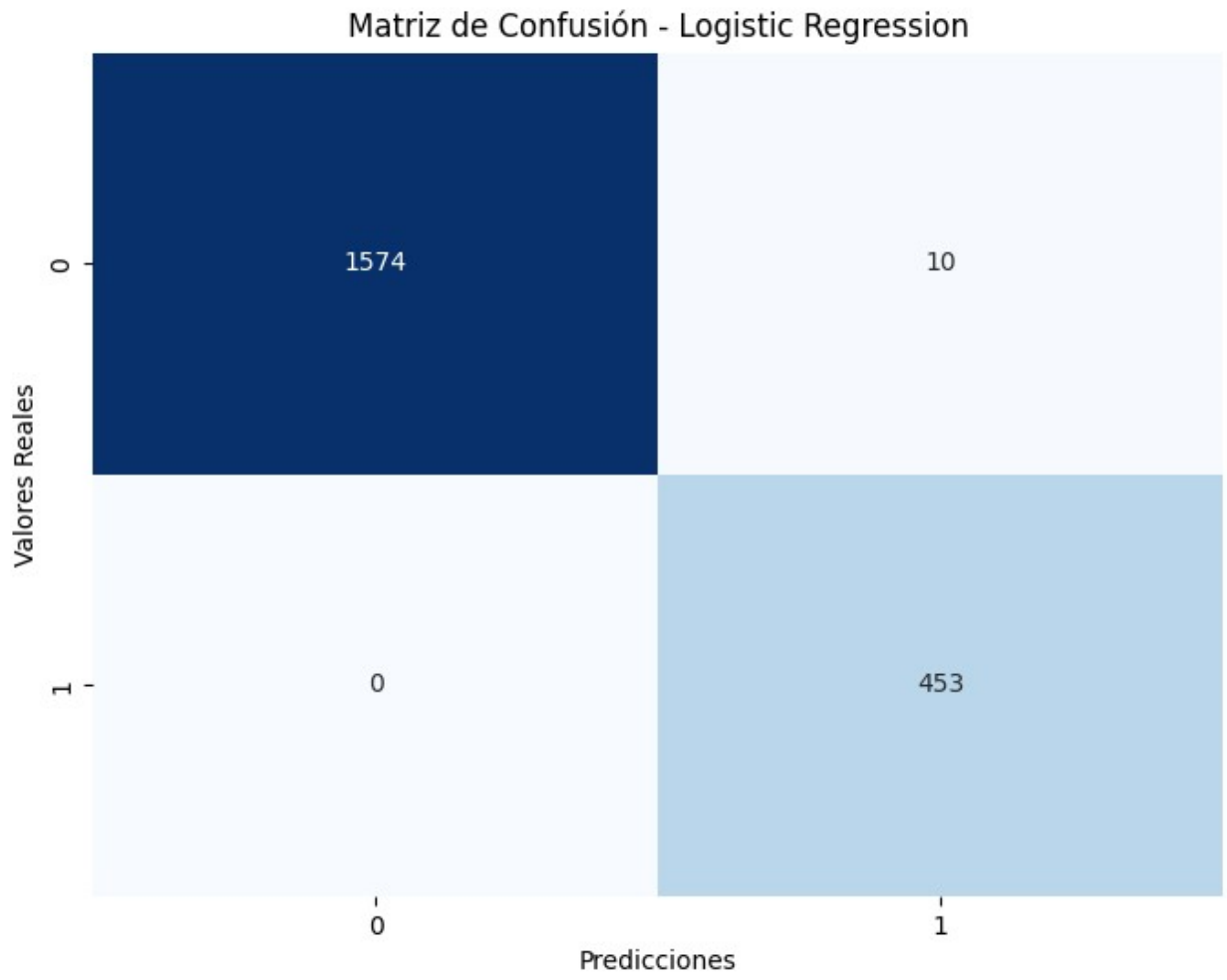
```
classification_rep = classification_report(y_test, y_pred)
print("Reporte de Clasificación:")
print(classification_rep)
```

```
Reporte de Clasificación:
              precision    recall  f1-score   support

           0       1.00      0.99      1.00      1584
           1       0.98      1.00      0.99       453

    accuracy                           1.00      2037
   macro avg       0.99      1.00      0.99      2037
weighted avg       1.00      1.00      1.00      2037
```

Se crea y muestra la confusion matrix

The confusion matrix is created and displayed.

```
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
cbar=False)
plt.xlabel('Predicciones')
plt.ylabel('Valores Reales')
plt.title('Matriz de Confusión - Logistic Regression')
plt.show()
```



Graficar la curva ROC

Plot the ROC curve

```
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f'Logistic Regression (AUC =
{roc_auc_LogisticRegression:.2f})', color='blue')
plt.plot([0, 1], [0, 1], 'r--')
plt.xlabel('Tasa de Falsos Positivos (FPR)')
plt.ylabel('Tasa de Verdaderos Positivos (TPR)')
plt.title('Curva ROC - Logistic Regression')
plt.legend(loc='lower right')
plt.show()
```

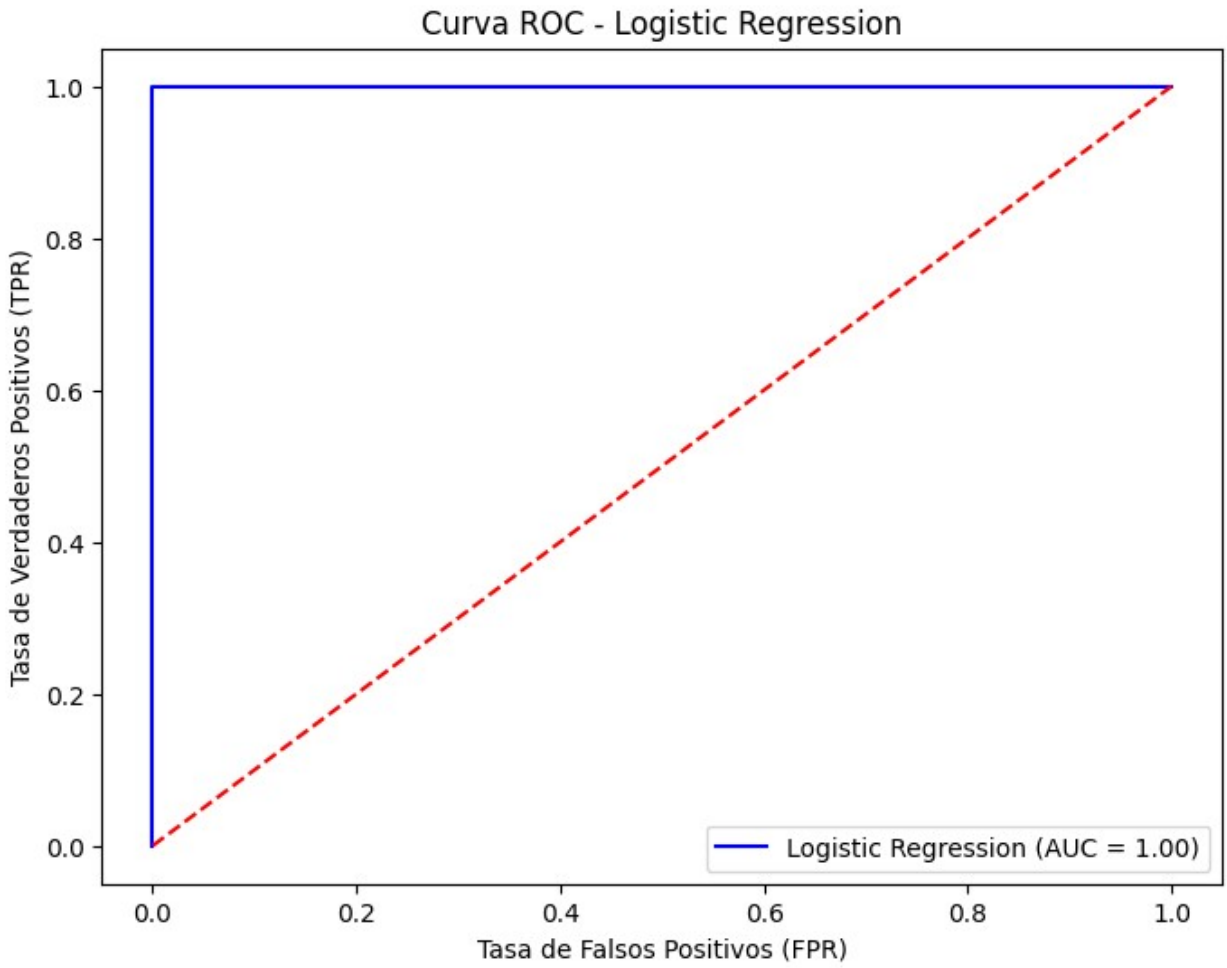
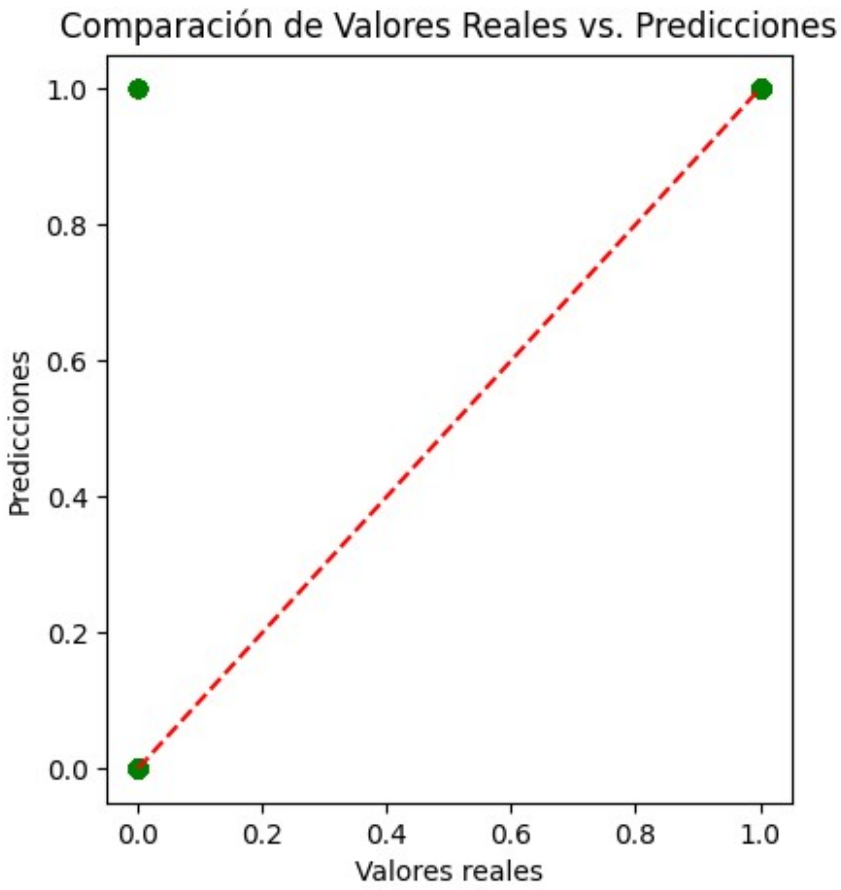


Gráfico y comparación de valores reales con las predicciones

Graph and comparison of actual values with predictions

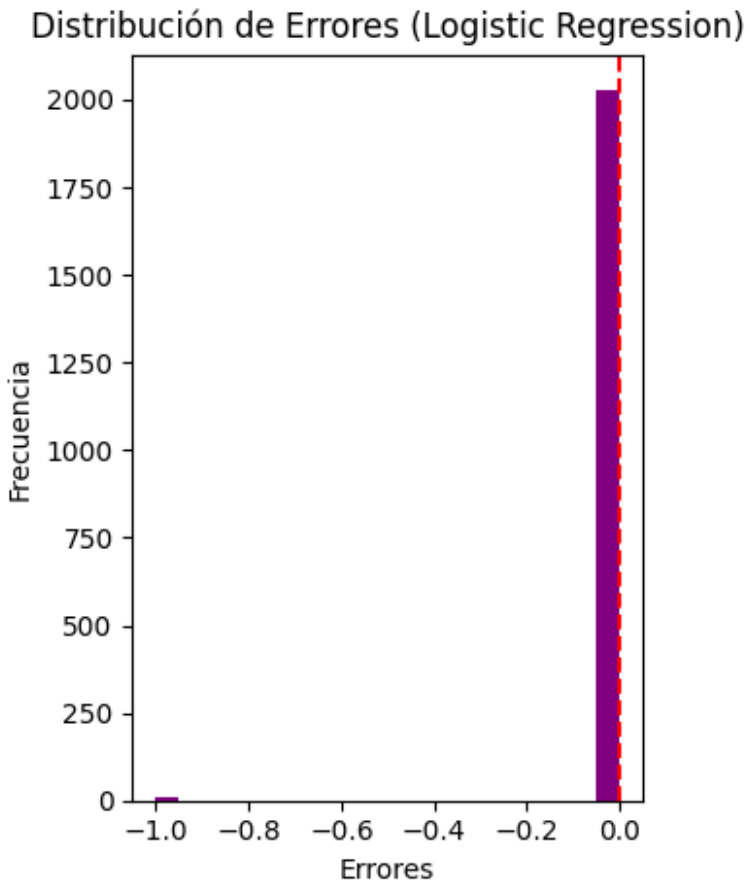
```
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.scatter(y_test, y_pred, color='green')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], '--',
color='red')
plt.xlabel('Valores reales')
plt.ylabel('Predicciones')
plt.title('Comparación de Valores Reales vs. Predicciones')

Text(0.5, 1.0, 'Comparación de Valores Reales vs. Predicciones')
```



Distribución de errores

Error distribution

```
residuals = y_test - y_pred
plt.subplot(1, 2, 2)
plt.hist(residuals, bins=20, color='purple')
plt.axvline(0, color='red', linestyle='--')
plt.xlabel('Errores')
plt.ylabel('Frecuencia')
plt.title('Distribución de Errores (Logistic Regression)')
plt.tight_layout()
plt.show()
```



DecisionTreeClassifier

Realizar una búsqueda de hiperparámetros usando GridSearchCV y encontrar cual es el mejor valor para el parámetro, almacenandolo en una variable para ocuparlo en el modelo

Perform a hyperparameter search using GridSearchCV and find the best value for the parameter, storing it in a variable to use it in the model.

```
param_grid = {'max_depth': [3, 5, 10, None]}
grid_search = GridSearchCV(DecisionTreeClassifier(criterion='gini',
random_state=42), param_grid, cv=3)
grid_search.fit(X_train, y_train)
best_max_depth = grid_search.best_params_['max_depth']
print("Mejores parámetros:", grid_search.best_params_)

Mejores parámetros: {'max_depth': None}
```

Se crea, se entrena y se realizan las predicciones del modelo DecisionTreeClassifier

The DecisionTreeClassifier model is created, trained, and predictions are made.

```
tree_model = DecisionTreeClassifier(criterion='gini',
max_depth=best_max_depth, random_state=42)
tree_model.fit(X_train, y_train)
y_pred_prob_tree = tree_model.predict_proba(X_test)[:, 1]
y_pred_tree = tree_model.predict(X_test)
y_pred_train_tree = tree_model.predict(X_train)
y_pred_prob_train_tree = tree_model.predict_proba(X_train)[:, 1]
```

Se calculan y muestran las métricas de rendimiento para el modelo LogisticRegression y las métricas del conjunto de pruebas para identificar si existe overfitting

The performance metrics for the LogisticRegression model and the metrics for the test set are calculated and displayed to identify if there is overfitting.

```
accuracy_DecisionTree = accuracy_score(y_test, y_pred_tree)
precision_DecisionTree = precision_score(y_test, y_pred_tree)
recall_DecisionTree = recall_score(y_test, y_pred_tree)
f1_DecisionTree = f1_score(y_test, y_pred_tree)
roc_auc_DecisionTree = roc_auc_score(y_test, y_pred_prob_tree)

accuracy_train_DecisionTree = accuracy_score(y_train,
y_pred_train_tree)
precision_train_DecisionTree = precision_score(y_train,
y_pred_train_tree)
recall_train_DecisionTree = recall_score(y_train, y_pred_train_tree)
f1_train_DecisionTree = f1_score(y_train, y_pred_train_tree)
roc_auc_train_DecisionTree = roc_auc_score(y_train,
y_pred_prob_train_tree)

print("Métricas en el conjunto de entrenamiento:")
print(f"Accuracy: {accuracy_train_DecisionTree:.4f}")
print(f"Precision: {precision_train_DecisionTree:.4f}")
print(f"Recall: {recall_train_DecisionTree:.4f}")
print(f"F1-Score: {f1_train_DecisionTree:.4f}")
print(f"AUC-ROC: {roc_auc_train_DecisionTree:.4f}")

print("\nMétricas en el conjunto de prueba:")
print(f"Accuracy: {accuracy_DecisionTree:.4f}")
print(f"Precision: {precision_DecisionTree:.4f}")
print(f"Recall: {recall_DecisionTree:.4f}")
print(f"F1-Score: {f1_DecisionTree:.4f}")
print(f"AUC-ROC: {roc_auc_DecisionTree:.4f}")
```

```
Métricas en el conjunto de entrenamiento:
Accuracy: 1.0000
Precision: 1.0000
Recall: 1.0000
F1-Score: 1.0000
AUC-ROC: 1.0000
```

```
Métricas en el conjunto de prueba:
Accuracy: 0.9720
Precision: 0.9304
Recall: 0.9448
F1-Score: 0.9376
AUC-ROC: 0.9623
```

Reporte de clasificación

Classification report

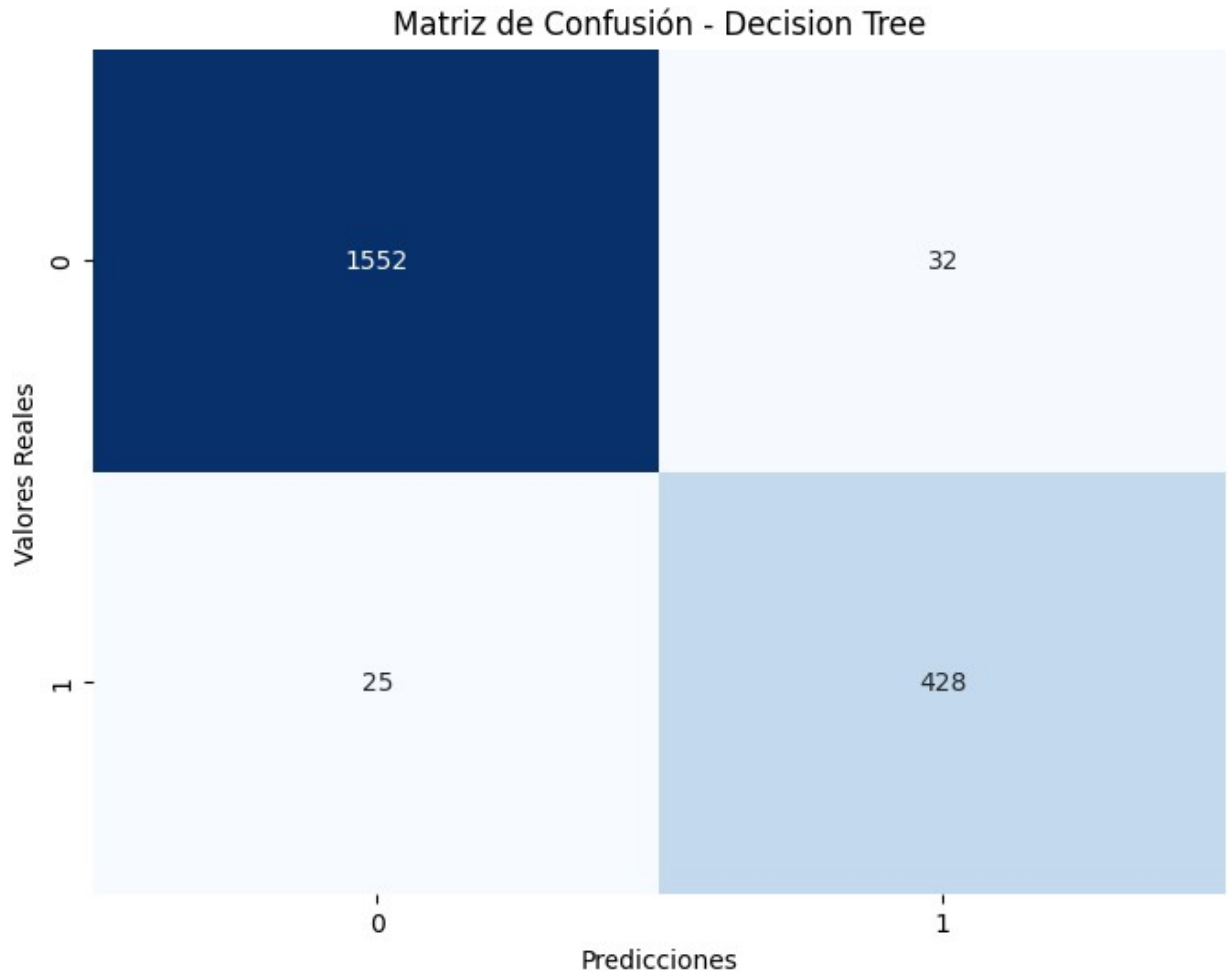
```
classification_rep_tree = classification_report(y_test, y_pred_tree)
print("Reporte de Clasificación:")
print(classification_rep_tree)
```

```
Reporte de Clasificación:
              precision    recall  f1-score   support

           0       0.98      0.98      0.98      1584
           1       0.93      0.94      0.94       453

    accuracy                           0.97      2037
   macro avg       0.96      0.96      0.96      2037
weighted avg       0.97      0.97      0.97      2037
```

Matriz de Confusión

Confusion Matrix

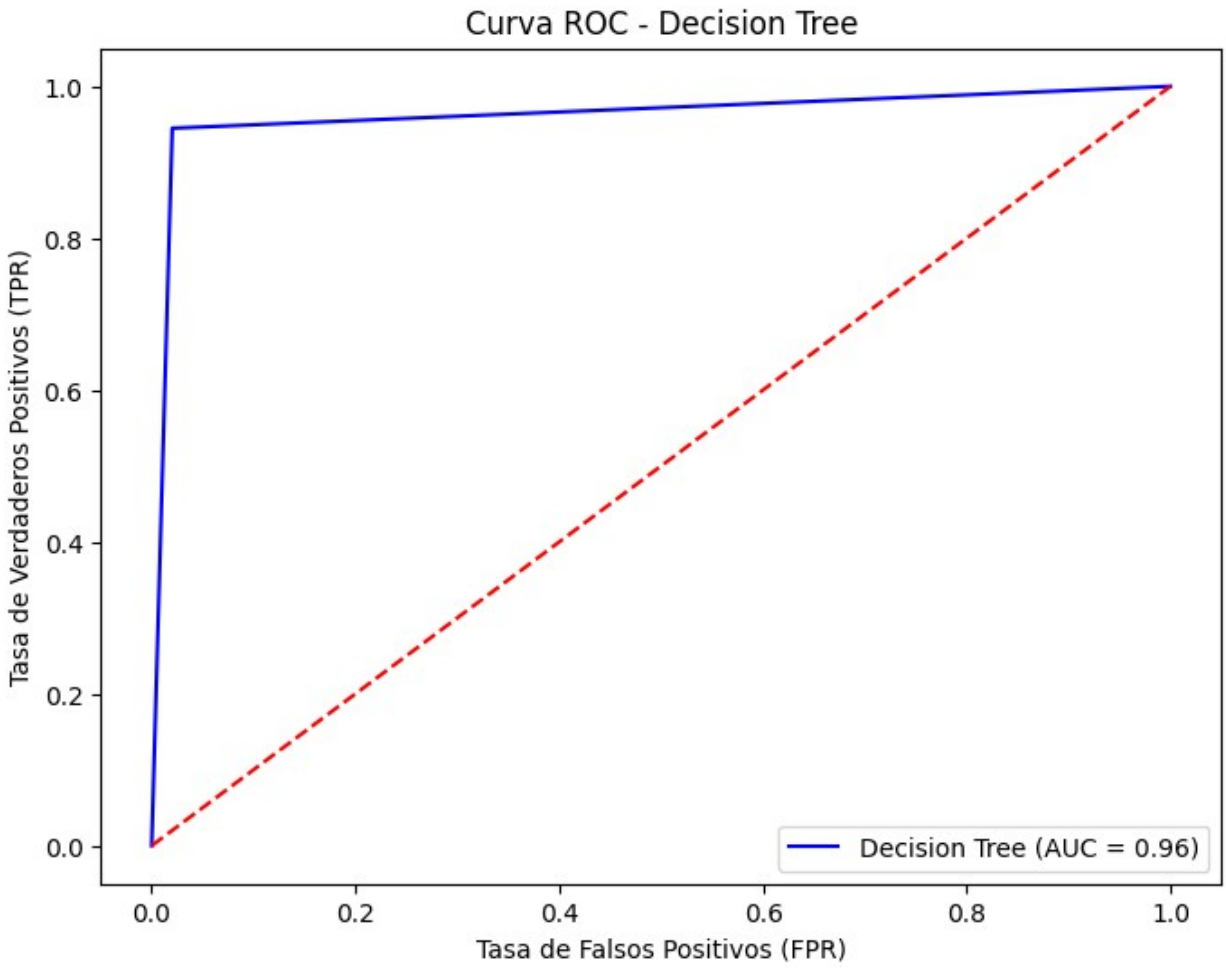
```
conf_matrix_tree = confusion_matrix(y_test, y_pred_tree)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_tree, annot=True, fmt='d', cmap='Blues',
cbar=False)
plt.xlabel('Predicciones')
plt.ylabel('Valores Reales')
plt.title('Matriz de Confusión - Decision Tree')
plt.show()
```



Curva ROC

ROC Curve

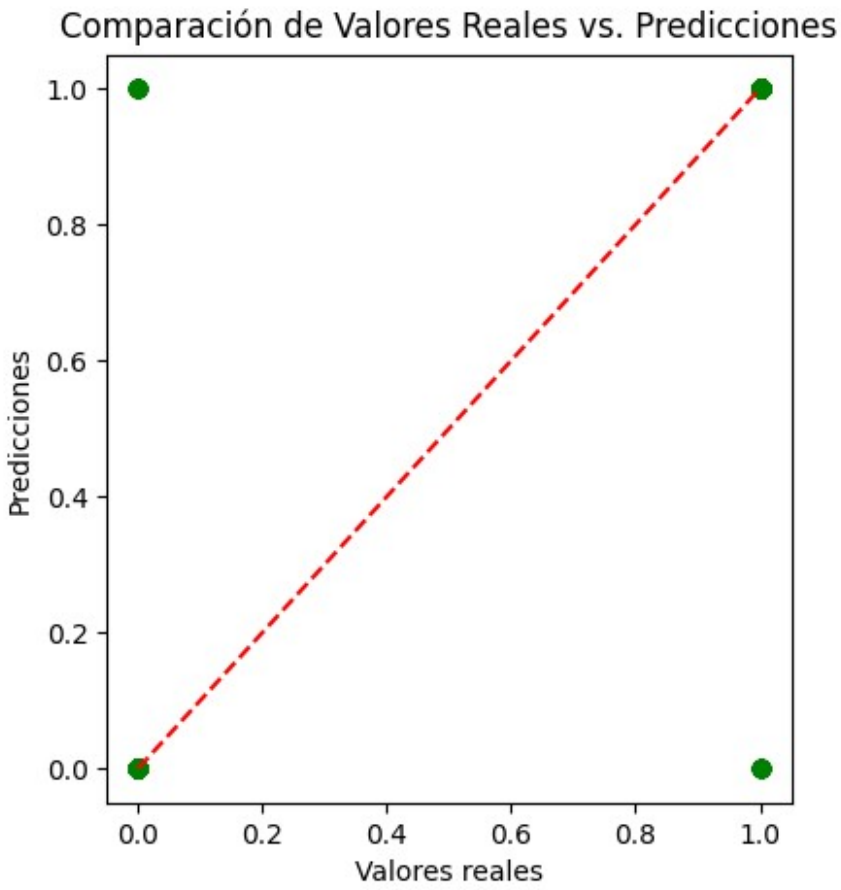
```
fpr_tree, tpr_tree, thresholds_tree = roc_curve(y_test,
y_pred_prob_tree)
plt.figure(figsize=(8, 6))
plt.plot(fpr_tree, tpr_tree, label=f'Decision Tree (AUC =
{roc_auc_DecisionTree:.2f})', color='blue')
plt.plot([0, 1], [0, 1], 'r--')
plt.xlabel('Tasa de Falsos Positivos (FPR)')
plt.ylabel('Tasa de Verdaderos Positivos (TPR)')
plt.title('Curva ROC - Decision Tree')
plt.legend(loc='lower right')
plt.show()
```



Comparación de valores reales vs predicciones

Comparison of actual values vs predictions

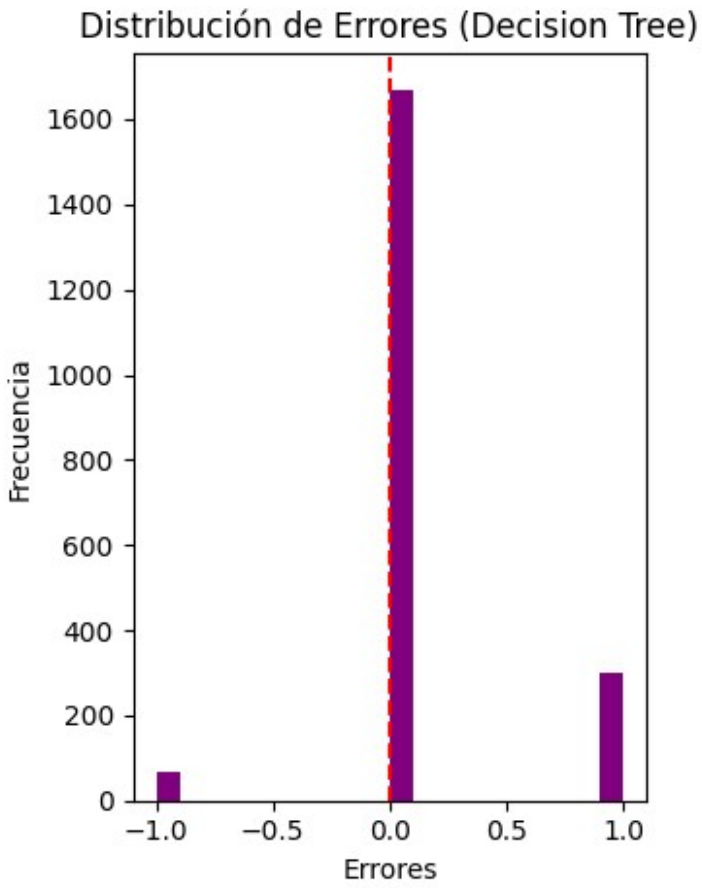
```
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.scatter(y_test, y_pred_tree, color='green')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], '--',
color='red')
plt.xlabel('Valores reales')
plt.ylabel('Predicciones')
plt.title('Comparación de Valores Reales vs. Predicciones')

Text(0.5, 1.0, 'Comparación de Valores Reales vs. Predicciones')
```



Distribución de errores

Error distribution

```
residuals_tree = y_test - y_pred_tree
plt.subplot(1, 2, 2)
plt.hist(residuals_tree, bins=20, color='purple')
plt.axvline(0, color='red', linestyle='--')
plt.xlabel('Errores')
plt.ylabel('Frecuencia')
plt.title('Distribución de Errores (Decision Tree)')
plt.tight_layout()
plt.show()
```



RandomForestClassifier

Realizar una búsqueda de hiperparámetros usando GridSearchCV y encontrar cual es el mejor valor para el parámetro, almacenandolo en una variable para ocuparlo en el modelo

Perform a hyperparameter search using GridSearchCV and find the best value for the parameter, storing it in a variable to use it in the model.

```
param_grid = {'n_estimators': [50, 100, 200]}
grid_search = GridSearchCV(RandomForestClassifier(random_state=42),
param_grid, cv=3)
grid_search.fit(X_train, y_train)
best_n_estimators = grid_search.best_params_['n_estimators']
print("Mejores parámetros:", grid_search.best_params_)
```

```
Mejores parámetros: {'n_estimators': 50}
```

Se crea, se entrena y se realizan las predicciones del modelo RandomForestClassifier

The RandomForestClassifier model is created, trained, and predictions are made.

```
rf_model = RandomForestClassifier(n_estimators=best_n_estimators,
random_state=42)
rf_model.fit(X_train, y_train)
y_pred_prob_rf = rf_model.predict_proba(X_test)[:, 1]
y_pred_rf = rf_model.predict(X_test)
y_pred_train_rf = rf_model.predict(X_train)
y_pred_prob_train_rf = rf_model.predict_proba(X_train)[:, 1]
```

Se calculan y muestran las métricas de rendimiento para el modelo LogisticRegression y las métricas del conjunto de pruebas para identificar si existe overfitting.

The performance metrics for the LogisticRegression model and the metrics for the test set are calculated and displayed to identify if there is overfitting.

```
accuracy_RandomForest = accuracy_score(y_test, y_pred_rf)
precision_RandomForest = precision_score(y_test, y_pred_rf)
recall_RandomForest = recall_score(y_test, y_pred_rf)
f1_RandomForest = f1_score(y_test, y_pred_rf)
roc_auc_RandomForest = roc_auc_score(y_test, y_pred_prob_rf)

accuracy_train_RandomForest = accuracy_score(y_train, y_pred_train_rf)
precision_train_RandomForest = precision_score(y_train,
y_pred_train_rf)
recall_train_RandomForest = recall_score(y_train, y_pred_train_rf)
f1_train_RandomForest = f1_score(y_train, y_pred_train_rf)
roc_auc_train_RandomForest = roc_auc_score(y_train,
y_pred_prob_train_rf)

print("Métricas en el conjunto de entrenamiento:")
print(f"Accuracy: {accuracy_train_RandomForest:.4f}")
print(f"Precision: {precision_train_RandomForest:.4f}")
print(f"Recall: {recall_train_RandomForest:.4f}")
print(f"F1-Score: {f1_train_RandomForest:.4f}")
print(f"AUC-ROC: {roc_auc_train_RandomForest:.4f}")

print("\nMétricas en el conjunto de prueba:")
print(f"Accuracy: {accuracy_RandomForest:.4f}")
print(f"Precision: {precision_RandomForest:.4f}")
print(f"Recall: {recall_RandomForest:.4f}")
print(f"F1-Score: {f1_RandomForest:.4f}")
print(f"AUC-ROC: {roc_auc_RandomForest:.4f}")
```

```
Métricas en el conjunto de entrenamiento:
Accuracy: 1.0000
Precision: 1.0000
Recall: 1.0000
F1-Score: 1.0000
AUC-ROC: 1.0000

Métricas en el conjunto de prueba:
```

```
Accuracy: 0.9666
Precision: 0.9508
Recall: 0.8962
F1-Score: 0.9227
AUC-ROC: 0.9958
```

Reporte de clasificación

Classification report

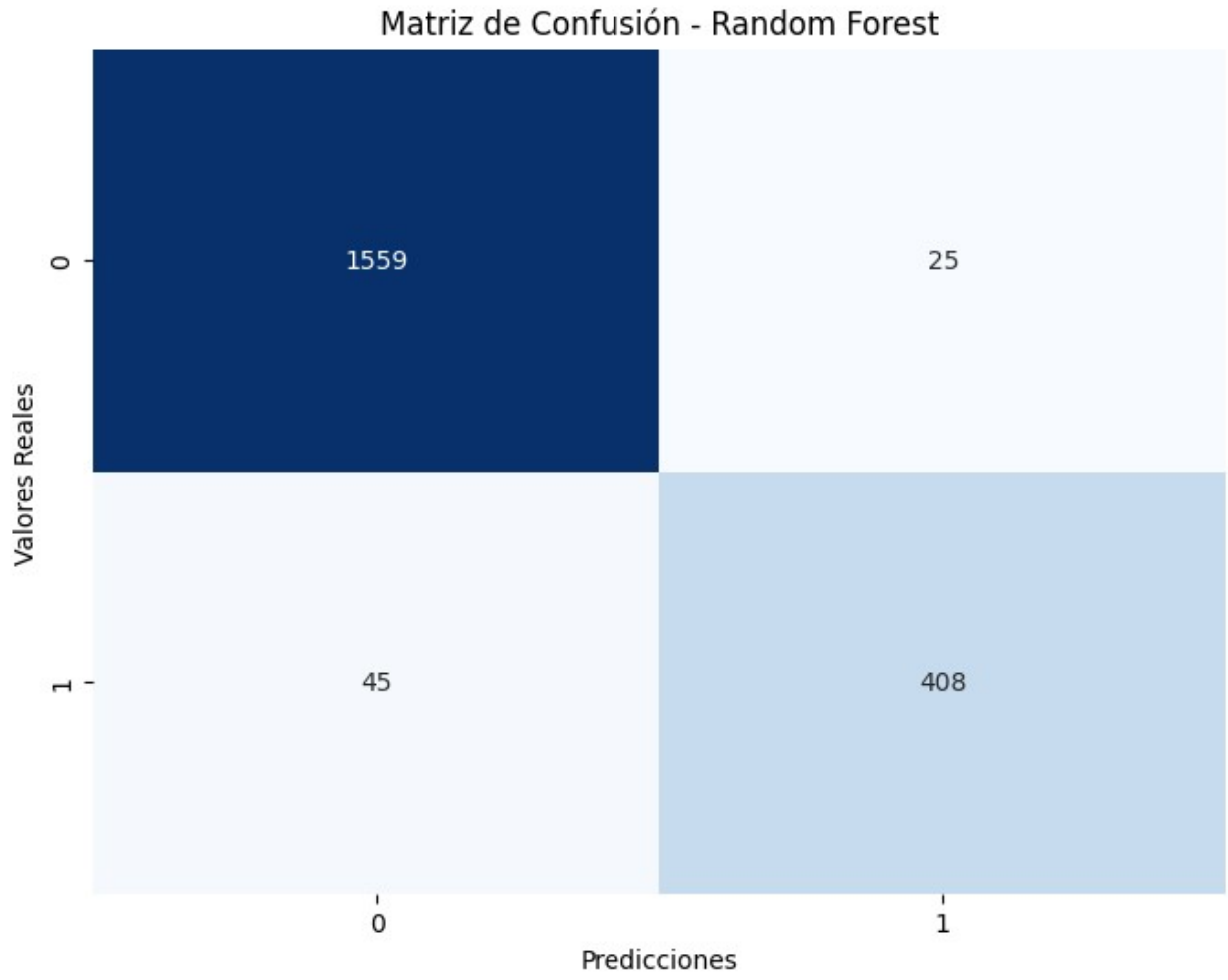
```
classification_rep_rf = classification_report(y_test, y_pred_rf)
print("Reporte de Clasificación:")
print(classification_rep_rf)
```

```
Reporte de Clasificación:
              precision    recall  f1-score   support

           0       0.97      0.98      0.98      1584
           1       0.94      0.90      0.92       453

    accuracy                           0.97      2037
   macro avg       0.96      0.94      0.95      2037
weighted avg       0.97      0.97      0.97      2037
```

Matriz de confusión

Confusion Matrix

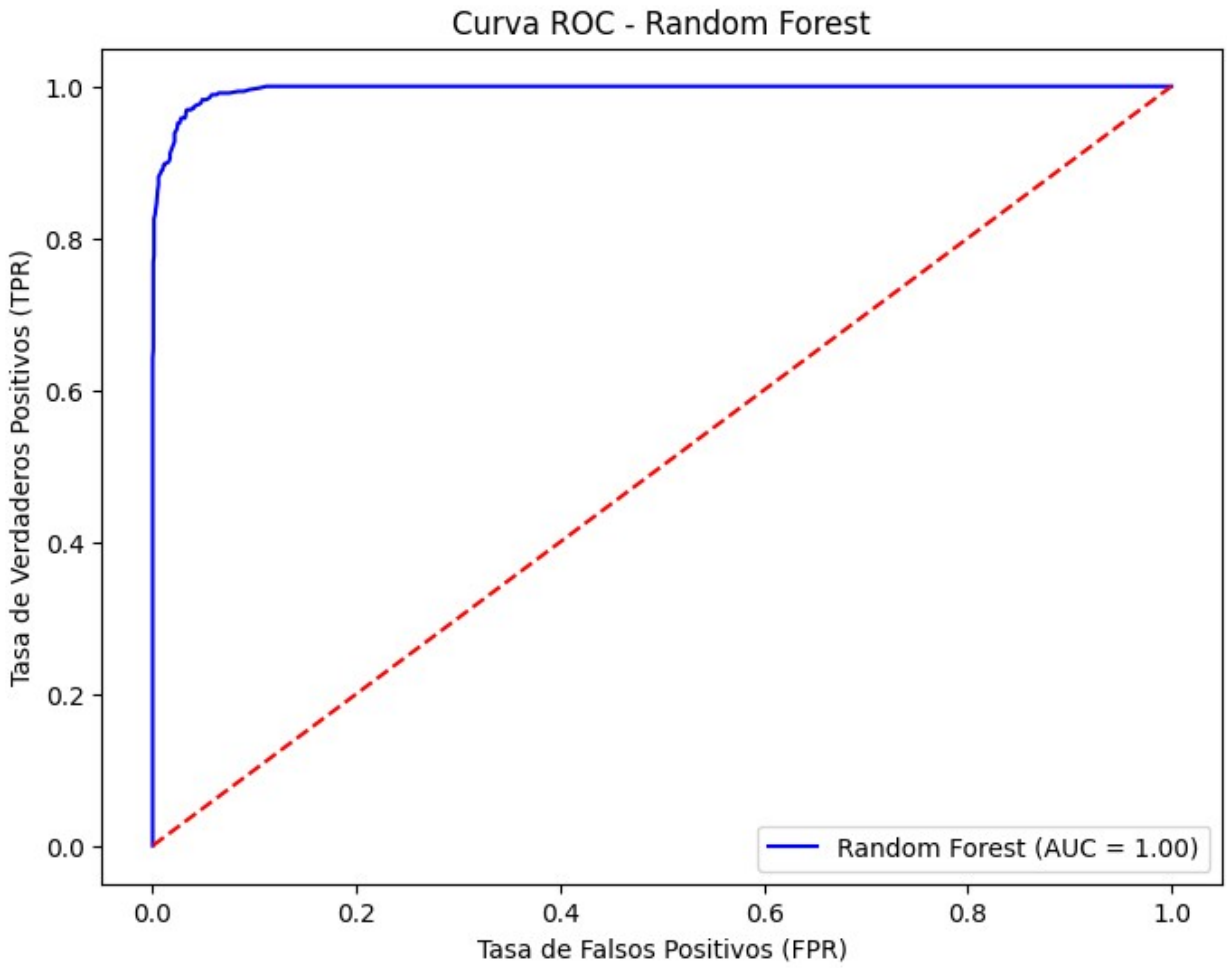
```
conf_matrix_rf = confusion_matrix(y_test, y_pred_rf)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_rf, annot=True, fmt='d', cmap='Blues',
cbar=False)
plt.xlabel('Predicciones')
plt.ylabel('Valores Reales')
plt.title('Matriz de Confusión - Random Forest')
plt.show()
```



Curva ROC

ROC Curve

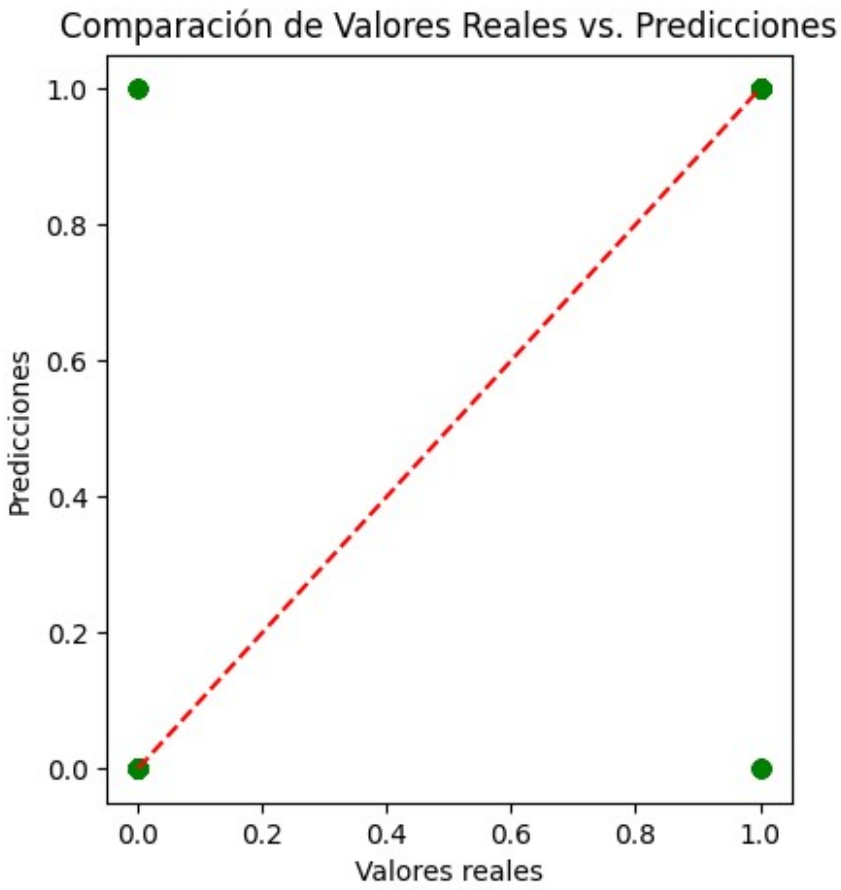
```
fpr_rf, tpr_rf, thresholds_rf = roc_curve(y_test, y_pred_prob_rf)
plt.figure(figsize=(8, 6))
plt.plot(fpr_rf, tpr_rf, label=f'Random Forest (AUC =
{roc_auc_RandomForest:.2f})', color='blue')
plt.plot([0, 1], [0, 1], 'r--')
plt.xlabel('Tasa de Falsos Positivos (FPR)')
plt.ylabel('Tasa de Verdaderos Positivos (TPR)')
plt.title('Curva ROC - Random Forest')
plt.legend(loc='lower right')
plt.show()
```



Comparación de valores reales vs predicciones

Comparison of actual values vs predictions

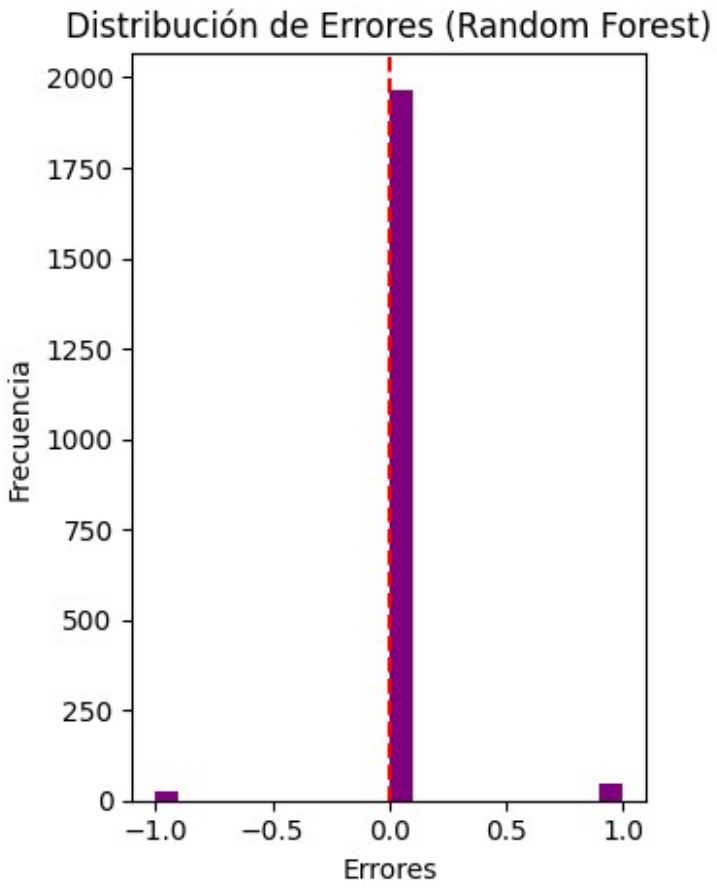
```
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.scatter(y_test, y_pred_rf, color='green')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], '--',
color='red')
plt.xlabel('Valores reales')
plt.ylabel('Predicciones')
plt.title('Comparación de Valores Reales vs. Predicciones')

Text(0.5, 1.0, 'Comparación de Valores Reales vs. Predicciones')
```



Distribución de errores

Error distribution

```
residuals_rf = y_test - y_pred_rf
plt.subplot(1, 2, 2)
plt.hist(residuals_rf, bins=20, color='purple')
plt.axvline(0, color='red', linestyle='--')
plt.xlabel('Errores')
plt.ylabel('Frecuencia')
plt.title('Distribución de Errores (Random Forest)')
plt.tight_layout()
plt.show()
```



**SVC (Support Vector Classifier)*

Realizar una búsqueda de hiperparámetros usando GridSearchCV y encontrar cual es el mejor valor para el parámetro, almacenandolo en una variable para ocuparlo en el modelo

Perform a hyperparameter search using GridSearchCV and find the best value for the parameter, storing it in a variable to use it in the model.

```
param_grid = {'C': [0.1, 1, 10]}
grid_search = GridSearchCV(SVC(kernel='linear', probability=True,
random_state=42), param_grid, cv=3)
grid_search.fit(X_train, y_train)
print("Mejores parámetros:", grid_search.best_params_)

best_C = grid_search.best_params_['C']
```

```
Mejores parámetros: {'C': 1}
```

Se crea, se entrena y se realizan las predicciones del modelo SVC

The SVC model is created, trained, and predictions are made.

```
svc_model = SVC(kernel='linear', C=best_C, probability=True,
random_state=42)
svc_model.fit(X_train, y_train)
y_pred_prob_svc = svc_model.predict_proba(X_test)[:, 1]
y_pred_svc = svc_model.predict(X_test)

y_pred_train_svc = svc_model.predict(X_train)
y_pred_prob_train_svc = svc_model.predict_proba(X_train)[:, 1]
```

Se calculan y muestran las métricas de rendimiento para el modelo SVC y las métricas del conjunto de pruebas para identificar si existe overfitting

The performance metrics for the SVC model and the test set metrics are calculated and displayed to identify if overfitting exists.

```
accuracy_train_SVC = accuracy_score(y_train, y_pred_train_svc)
precision_train_SVC = precision_score(y_train, y_pred_train_svc)
recall_train_SVC = recall_score(y_train, y_pred_train_svc)
f1_train_SVC = f1_score(y_train, y_pred_train_svc)
roc_auc_train_SVC = roc_auc_score(y_train, y_pred_prob_train_svc)

accuracy_SVC = accuracy_score(y_test, y_pred_svc)
precision_SVC = precision_score(y_test, y_pred_svc)
recall_SVC = recall_score(y_test, y_pred_svc)
f1_SVC = f1_score(y_test, y_pred_svc)
roc_auc_SVC = roc_auc_score(y_test, y_pred_prob_svc)

print("Métricas en el conjunto de entrenamiento:")
print(f"Accuracy: {accuracy_train_SVC:.4f}")
print(f"Precision: {precision_train_SVC:.4f}")
print(f"Recall: {recall_train_SVC:.4f}")
print(f"F1-Score: {f1_train_SVC:.4f}")
print(f"AUC-ROC: {roc_auc_train_SVC:.4f}")

print("\nMétricas en el conjunto de prueba:")
print(f"Accuracy: {accuracy_SVC:.4f}")
print(f"Precision: {precision_SVC:.4f}")
print(f"Recall: {recall_SVC:.4f}")
print(f"F1-Score: {f1_SVC:.4f}")
print(f"AUC-ROC: {roc_auc_SVC:.4f}")
```

```
Métricas en el conjunto de entrenamiento:
Accuracy: 1.0000
Precision: 1.0000
Recall: 1.0000
F1-Score: 1.0000
AUC-ROC: 1.0000

Métricas en el conjunto de prueba:
Accuracy: 1.0000
```

```
Precision: 1.0000
Recall: 1.0000
F1-Score: 1.0000
AUC-ROC: 1.0000
```

Reporte de clasificación

Classification report

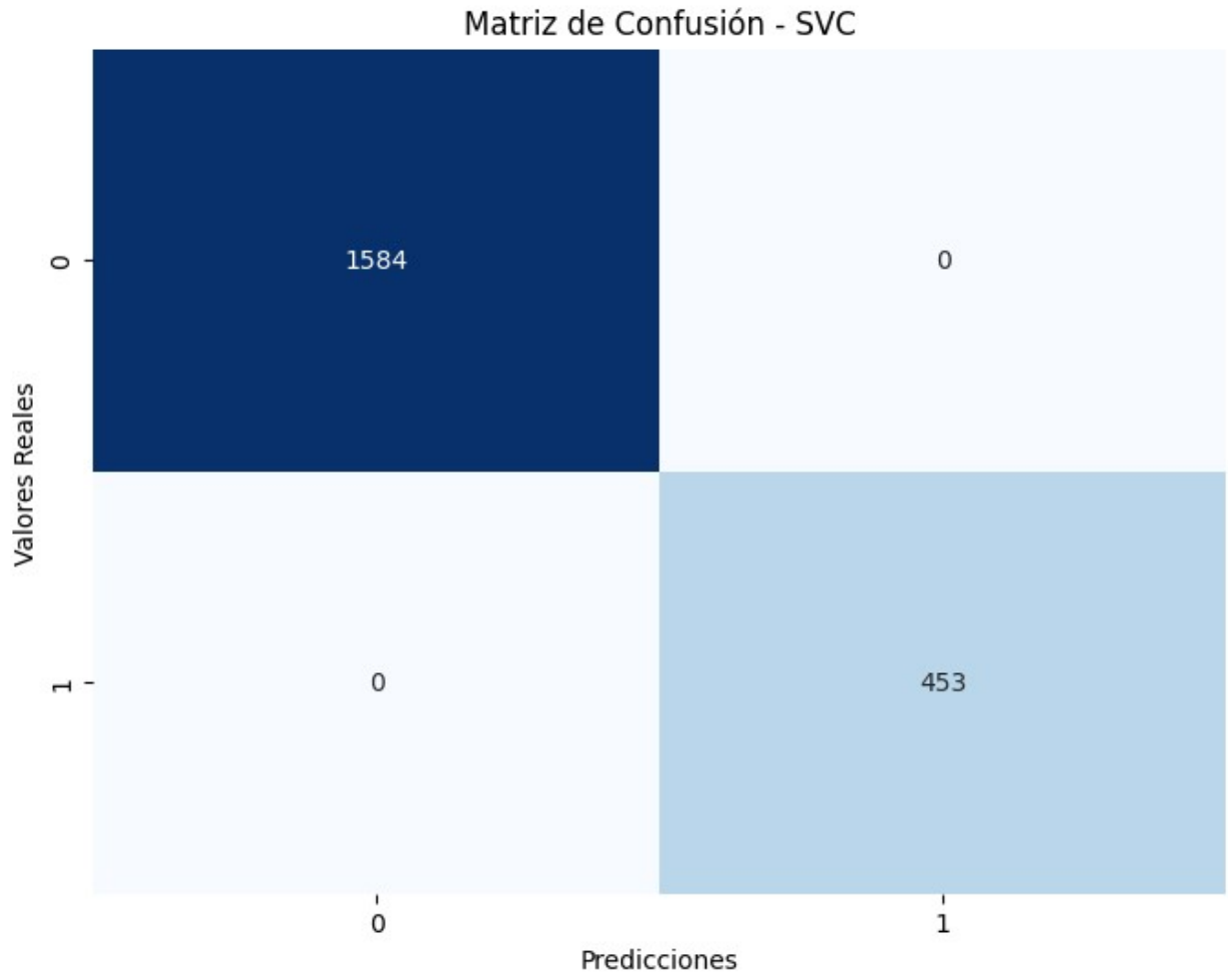
```
classification_rep_svc = classification_report(y_test, y_pred_svc)
print("Reporte de Clasificación:")
print(classification_rep_svc)
```

```
Reporte de Clasificación:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1584
           1       1.00      1.00      1.00       453

    accuracy                           1.00      2037
   macro avg       1.00      1.00      1.00      2037
weighted avg       1.00      1.00      1.00      2037
```

Matriz de confusión

Confusion Matrix

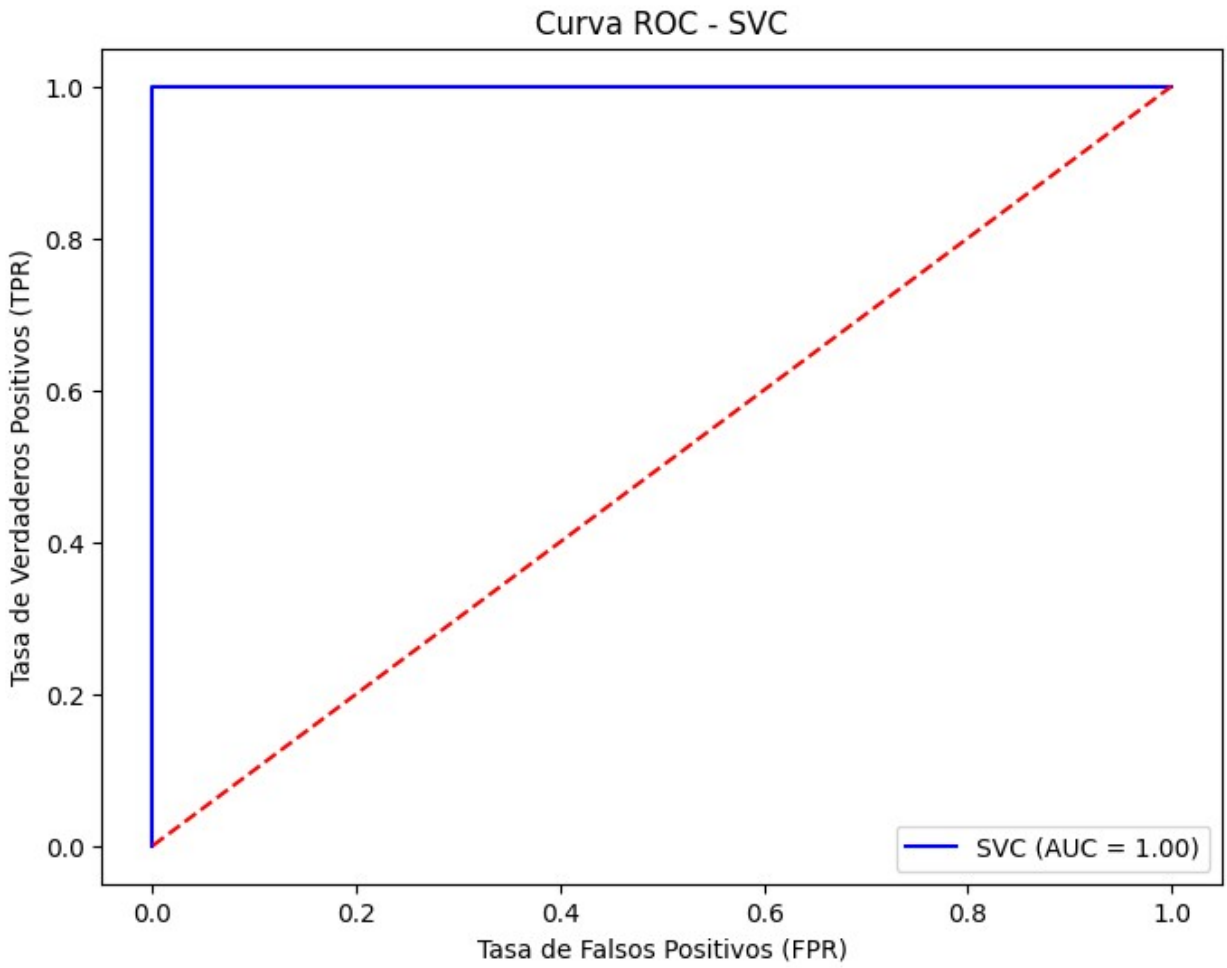
```
conf_matrix_svc = confusion_matrix(y_test, y_pred_svc)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_svc, annot=True, fmt='d', cmap='Blues',
cbar=False)
plt.xlabel('Predicciones')
plt.ylabel('Valores Reales')
plt.title('Matriz de Confusión - SVC')
plt.show()
```



Curva ROC

ROC Curve

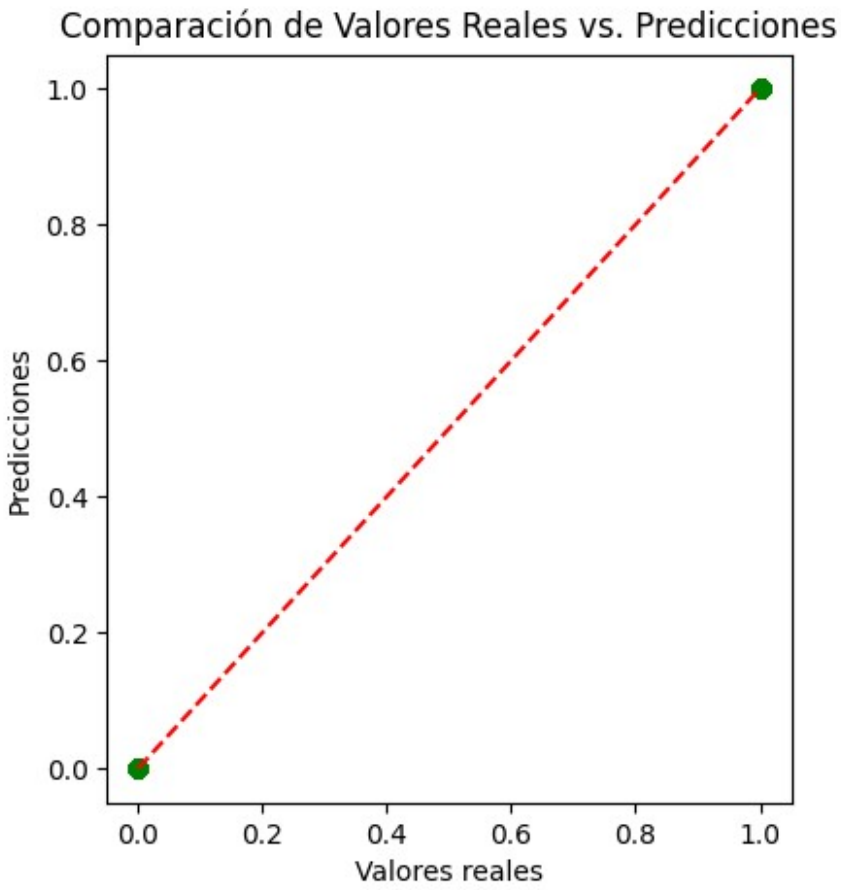
```
fpr_svc, tpr_svc, thresholds_svc = roc_curve(y_test, y_pred_prob_svc)
plt.figure(figsize=(8, 6))
plt.plot(fpr_svc, tpr_svc, label=f'SVC (AUC = {roc_auc_SVC:.2f})',
color='blue')
plt.plot([0, 1], [0, 1], 'r--')
plt.xlabel('Tasa de Falsos Positivos (FPR)')
plt.ylabel('Tasa de Verdaderos Positivos (TPR)')
plt.title('Curva ROC - SVC')
plt.legend(loc='lower right')
plt.show()
```



Comparación de valores reales vs predicciones

Comparison of actual values vs predictions

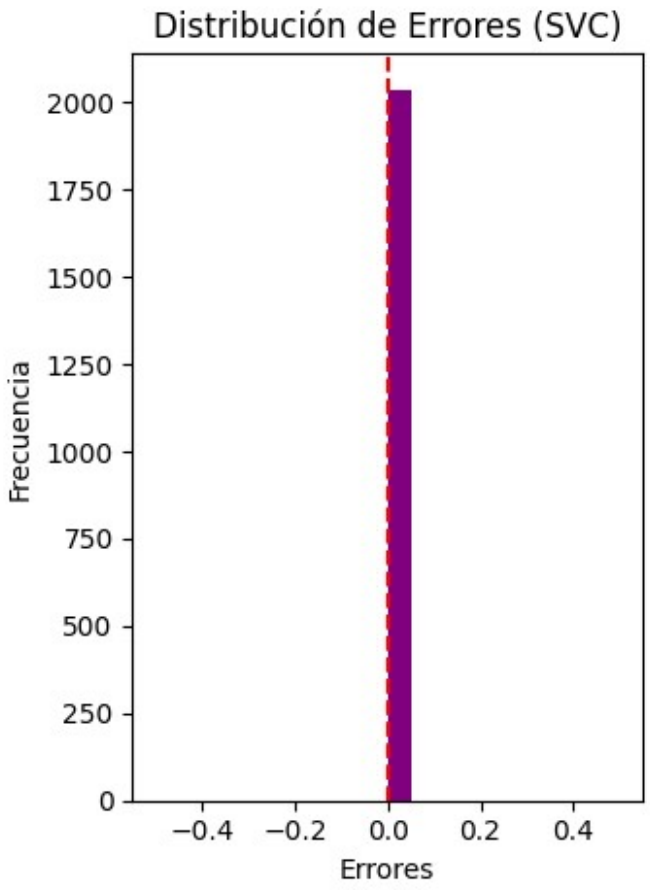
```
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.scatter(y_test, y_pred_svc, color='green')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], '--',
color='red')
plt.xlabel('Valores reales')
plt.ylabel('Predicciones')
plt.title('Comparación de Valores Reales vs. Predicciones')

Text(0.5, 1.0, 'Comparación de Valores Reales vs. Predicciones')
```



Distribución de errores

Error distribution

```
residuals_svc = y_test - y_pred_svc
plt.subplot(1, 2, 2)
plt.hist(residuals_svc, bins=20, color='purple')
plt.axvline(0, color='red', linestyle='--')
plt.xlabel('Errores')
plt.ylabel('Frecuencia')
plt.title('Distribución de Errores (SVC)')
plt.tight_layout()
plt.show()
```




```
# Insertar cuantos bloques de código consideren necesarios

# Realizar tarea de clasificación de datos orientado al caso entregado
```

Fase 5: Evaluation

Modelos de Regresión

```
print("Mean Absolute Error de KNeighborsRegressor (MAE):",
mae_KNeighborsRegressor)
print("Mean Absolute Error de RandomForestRegressor (MAE):",
mae_RandomForestRegressor)
```

```
Mean Absolute Error de KNeighborsRegressor (MAE): 0.023969706582109968
Mean Absolute Error de RandomForestRegressor (MAE):
0.002691546733640345
```

Con la MAE se puede obtener que modelo está ejecutando mejor los datos y comete menos errores en las predicciones, el cual es el modelo RandomForestRegressor, que tiene un error promedio absoluto significativamente más bajo que el de KNeighborsRegressor.

With the MAE, one can determine which model is performing better with the data and making fewer prediction errors, which is the RandomForestRegressor model, as it has a significantly lower mean absolute error than the KNeighborsRegressor.

```
print("Mean Squared Error de KNeighborsRegressor (MSE):",
mse_KNeighborsRegressor)
print("Mean Squared Error de RandomForestRegressor (MSE):",
mse_RandomForestRegressor)
```

```
Mean Squared Error de KNeighborsRegressor (MSE): 0.0020063518119173736
Mean Squared Error de RandomForestRegressor (MSE):
3.0972048547968354e-05
```

Con el MSE podemos obtener si se está ajustando mejor los datos y tiene una menor dispersión en los errores algún modelo, lo cual el modelo RandomForestRegressor sigue mostrando un rendimiento significativamente mejor, con un error cuadrático medio mucho más bajo que el del KNeighborsRegressor.

With the MSE, we can determine if a model is fitting the data better and has a lower dispersion in the errors, and the RandomForestRegressor model continues to show significantly better performance, with a much lower mean squared error than the KNeighborsRegressor.

```
print("Root Mean Squared Error de KNeighborsRegressor (RMSE):",
rmse_KNeighborsRegressor)
print("Root Mean Squared Error de RandomForestRegressor (RMSE):",
rmse_RandomForestRegressor)
```

```
Root Mean Squared Error de KNeighborsRegressor (RMSE):
0.0447923186709214
Root Mean Squared Error de RandomForestRegressor (RMSE):
0.005565253682265379
```

Con el RMSE podemos obtener si las predicciones de un modelo están más cerca de los valores reales en promedio, en la cual el RandomForestRegressor muestra un desempeño mucho mejor, con un RMSE significativamente menor.

With the RMSE, we can determine if a model's predictions are closer to the actual values on average, in which the RandomForestRegressor shows much better performance, with a significantly lower RMSE.

```
print("Score del modelo de KNeighborsRegressor (R²):",
r2_KNeighborsRegressor)
print("Score del modelo de RandomForestRegressor (R²):",
r2_RandomForestRegressor)
```

```
Score del modelo de KNeighborsRegressor (R²): 0.9082273827382688
Score del modelo de RandomForestRegressor (R²): 0.9985833063073379
```

Con el R^2 podemos obtener la variabilidad en los datos, teniendo el RandomForestRegressor que un 99.86% de la variabilidad en los datos se explica por el modelo.

With R^2, we can obtain the variability in the data, with the RandomForestRegressor explaining 99.86% of the variability in the data.

Conclusión

El RandomForestRegressor no solo tiene un RMSE y MAE más bajos, sino que también explica una proporción mucho mayor de la variabilidad en los datos en comparación con el KNeighborsRegressor. Por lo cual es correcto afirmar que el Random Forest es el mejor modelo de regresión para el conjunto de datos

The RandomForestRegressor not only has lower RMSE and MAE, but it also explains a much greater proportion of the variability in the data compared to the KNeighborsRegressor. Therefore, it is correct to state that Random Forest is the best regression model for the dataset.

Modelos de Clasificación

```
print(f"Accuracy de LogisticRegression:
{accuracy_LogisticRegression:.4f}")
print(f"Accuracy de DecisionTree: {accuracy_DecisionTree:.4f}")
print(f"Accuracy de RandomForest: {accuracy_RandomForest:.4f}")
print(f"Accuracy de SVC: {accuracy_SVC:.4f}")
```

```
Accuracy de LogisticRegression: 0.9951
Accuracy de DecisionTree: 0.8198
Accuracy de RandomForest: 0.9656
Accuracy de SVC: 1.0000
```

El accuracy mide la proporción de predicciones correctas en los modelos de clasificación, tenemos que el modelo SVC tiene una precisión perfecta en la predicciones correctas.

Accuracy measures the proportion of correct predictions in classification models; we find that the SVC model has perfect accuracy in correct predictions.

```
print(f"Precision de LogisticRegression:
{precision_LogisticRegression:.4f}")
print(f"Precision de DecisionTree: {precision_DecisionTree:.4f}")
print(f"Precision de RandomForest: {precision_RandomForest:.4f}")
print(f"Precision de SVC: {precision_SVC:.4f}")
```

```
Precision de LogisticRegression: 0.9784
Precision de DecisionTree: 0.6955
Precision de RandomForest: 0.9423
Precision de SVC: 1.0000
```

Con la precisión se puede evaluar el rendimiento de los modelos de clasificación, en este caso el SVC es el mejor modelo con un rendimiento perfecto a momento de evaluar el modelo.

With precision, the performance of classification models can be evaluated; in this case, the SVC is the best model with perfect performance at the time of model evaluation.

```
print(f"Recall de LogisticRegression:
{recall_LogisticRegression:.4f}")
print(f"Recall de DecisionTree: {recall_DecisionTree:.4f}")
print(f"Recall de RandomForest: {recall_RandomForest:.4f}")
print(f"Recall de SVC: {recall_SVC:.4f}")
```

```
Recall de LogisticRegression: 1.0000
Recall de DecisionTree: 0.3377
Recall de RandomForest: 0.9007
Recall de SVC: 1.0000
```

Con el Recall se puede obtener la proporción de verdaderos positivos sobre el total de instancias que realmente son positivas, donde dos modelos tienen un rendimiento perfecto, estos son el modelo de Regresión Logistica y el de SVC.

With Recall, you can obtain the proportion of true positives over the total instances that are actually positive, where two models perform perfectly, these are the Logistic Regression model and the SVC model.

```
print(f"F1-Score de LogisticRegression: {f1_LogisticRegression:.4f}")
print(f"F1-Score de DecisionTree: {f1_DecisionTree:.4f}")
print(f"F1-Score de RandomForest: {f1_RandomForest:.4f}")
print(f"F1-Score de SVC: {f1_SVC:.4f}")
```

```
F1-Score de LogisticRegression: 0.9891
F1-Score de DecisionTree: 0.4547
F1-Score de RandomForest: 0.9210
F1-Score de SVC: 1.0000
```

El F1-Score combina la precisión y el recall en un solo número, proporcionando una medida balanceada del rendimiento de un modelo de clasificación, nuevamente el modelo SVC tiene una puntuación perfecta en esta métrica.

The F1-Score combines precision and recall into a single number, providing a balanced measure of a classification model's performance; once again, the SVC model has a perfect score on this metric.

```
print(f"AUC-ROC de LogisticRegression:
{roc_auc_LogisticRegression:.4f}")
print(f"AUC-ROC de DecisionTree: {roc_auc_DecisionTree:.4f}")
print(f"AUC-ROC de RandomForest: {roc_auc_RandomForest:.4f}")
print(f"AUC-ROC de SVC: {roc_auc_SVC:.4f}")
```

```
AUC-ROC de LogisticRegression: 1.0000
AUC-ROC de DecisionTree: 0.7375
AUC-ROC de RandomForest: 0.9959
AUC-ROC de SVC: 1.0000
```

El AUC-ROC mide la capacidad de un modelo de clasificación para diferenciar entre clases positivas y negativas en diversos umbrales de decisión, nuevamente el SVC tiene una puntuación perfecta junto con el de Regresión Logistica.

The AUC-ROC measures the ability of a classification model to differentiate between positive and negative classes at various decision thresholds; once again, the SVC has a perfect score along with the Logistic Regression.

Conclusión

Tres de los cuatro modelos tienen puntuaciones muy buenas pero hay uno que tiene todas perfectas, este es el modelo SVC o Support Vector Classifier, donde basandonos en los resultados de las métricas de rendimiento, podemos obtener que el modelo está funcionando de manera perfecta con el conjunto de datos utilizado, alcanzando un rendimiento ideal en todas las métricas claves y otorgando un modelo eficiente para el manejo de datos.

Three of the four models have very good scores, but there is one that has perfect scores, which is the SVC or Support Vector Classifier model. Based on the performance metric results, we can determine that the model is functioning perfectly with the dataset used, achieving ideal performance in all key metrics and providing an efficient model for data handling.

```
# Insertar cuantos bloques de código consideren necesarios

# Evaluar con las métricas que corresponda los mejores modelos,
explicar cual es mejor para cada tarea y por qué
```

Fase 5: Deployment

```
# Insertar cuantos bloques de código consideren necesarios

# Realizar despliegue del modelo
```