Predicción del Torneo de Baloncesto NCAA 2017

Integrantes del grupo:

Andrés Darío Peña

Andrés Ramirez Aristizabal

Leider Caicedo Palacios

Materia:

Modelos y Simulación 1

Docente:

Raúl Ramos Pollán

Universidad de Antioquia

Facultad de ingeniería

Noviembre 2023

1. Describa el problema predictivo a resolver.

Predecir los resultados de un torneo de baloncesto universitario masculino en Estados Unidos en el año 2017. Utilizaremos los resultados recopilados de torneos pasados para poder construir y probar diferentes modelos de predicción. Después pronosticamos los resultados de todos los enfrentamientos posibles en el torneo de 2017.

2. Dataset que utilizaremos.

El dataset que vamos a utilizar es el de Machine Learning mania, un dataset basado en la competición de basketball NCAA del 2017. (link a continuación) https://www.kaggle.com/competitions/march-machine-learning-mania-2017/data?select=Regular_Season_Detailed_Results.csv, la cual posee 34 columnas con 76.600 instancias.

El archivo que contiene los datos se llama: ***RegularSeasonDetailedResults.csv.*** Este archivo contiene los resultados del juego por juego para 32 temporadas de datos históricos, de 1985 a 2015. Cada año, incluye todos los juegos jugados desde el día 0 hasta el 132.

"season" - Este es el año de la entrada asociada en seasons.csv (el año en que ocurre el torneo final)
"daynum" - Este número entero siempre varía de 0 a 132 y te dice en qué día se jugó el juego.
"wteam" - identifica el ID del equipo que ganó el juego.
"wscore" - identifica la cantidad de puntos anotados por el equipo ganador.
"lteam" - identifica el ID del equipo que perdió el juego.
"lscore" - identifica la cantidad de puntos anotados por el equipo perdedor.
"numot" - esto indica el número de períodos de tiempo extra en el juego, un número entero 0 o superior.

También las estadísticas totales a nivel de equipo para cada juego del equipo ganador.

wfgm - goles de campo realizados
wfga - intentos de goles de campo
wfgm3 - tres punteros hechos
wfga3 - intento de tres punteros
wftm - tiros libres hechos
wfta - intento de tiros libres
wor - rebotes ofensivos
wdr - rebotes defensivos
wast - asiste
wto - pérdidas de balón

wstl - roba
wblk - bloques
wpf - faltas personales

las estadísticas totales a nivel de equipo para cada juego del equipo perdedor:
Lfgm - goles de campo realizados
Lfga - intentos de goles de campo
Lfgm3 - tres punteros hechos
Lfga3 - intento de tres punteros
Lftm - tiros libres hechos
Lfta - intento de tiros libres
Lor - rebotes ofensivos
Ldr - rebotes defensivos
Last - asiste
Lto - pérdidas de balón
Lstl - roba
Lblk - bloques
Lpf - faltas personales

3. **Las métricas de desempeño requeridas (de machine learning y de negocio)**

La métrica de desempeño requerida en este caso es la **accuracy**. La accuracy es una métrica comúnmente utilizada en problemas de clasificación y mide la proporción de predicciones correctas respecto al total de predicciones realizadas. Se calcula como el número de predicciones correctas dividido por el total de predicciones.

En el contexto de este conjunto de datos de baloncesto universitario masculino, donde se busca predecir qué equipo ganará un juego, la accuracy nos permite medir la proporción de juegos que el modelo predice correctamente en comparación con el total de juegos en el conjunto de datos. En otras palabras, es una medida de la precisión general del modelo en predecir los resultados correctos.

El uso de accuracy en este conjunto de datos tiene sentido porque estamos interesados en saber cuántos de los enfrentamientos predichos por el modelo coinciden con el equipo ganador real. Para un torneo deportivo, especialmente uno donde la victoria es el objetivo final, saber con precisión qué equipo ganará es crucial.
La accuracy determina cuántas predicciones correctas hace el modelo en relación con todas las predicciones realizadas. En el contexto de este conjunto de datos, si el modelo predice correctamente qué equipo ganará un juego, esa predicción se considera un "acierto". La accuracy se calcula como:

$$Accuracy = \frac{Numero\ de\ predicciones\ correctas}{Total\ de\ predicciones}$$

4. **Un primer criterio sobre cuál sería el desempeño deseable en producción.**

Con este modelo esperamos predecir con un alto nivel de precisión el resultado de los partidos del torneo teniendo en cuenta resultados obtenidos en anteriores torneos. Esta información puede ser muy útil para realizar apuestas que permitan obtener una alto nivel de ganancias. Para que al apostador le resulte eficiente y le genere ganancias el uso del dataset el margen de error de este debe ser menor a 0.5.

Variable objetivo: "Wteam" se elige como variable objetivo porque representa el resultado central que queremos prever en el contexto del torneo de baloncesto universitario: el equipo que ganará en cada juego.

Simulación del dataset para requerimientos:

Se realizaron las simulaciones de los requerimientos pedidos para la base de datos seleccionada, los cuales fueron:

El 5% de los datos de al menos 3 columnas deben ser faltantes: Con este ciclo se ejecutó dicha simulación de datos faltantes en las columnas Lftm, Ldr y Lfga3

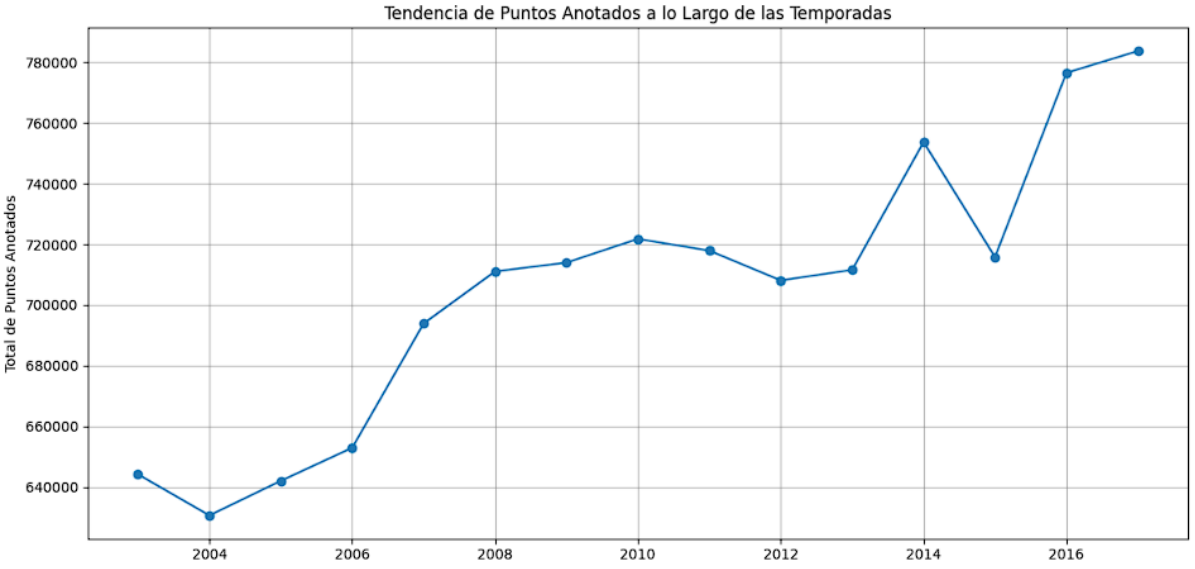
```
[ ]  for i in range(int(len(cdb1) * 0.05)):
         x = np.random.randint(len(cdb1))
         cdb1.loc[x, "Lftm"] = np.nan
         cdb1.loc[x, "Ldr"] = np.nan
         cdb1.loc[x, "Lfga3"] = np.nan
```

El resultado fue satisfactorio, donde el 5% de los datos de las columnas mencionadas (3744) se presentan como nulos gracias a la información proporcionada por la función isna de pandas dataframe.

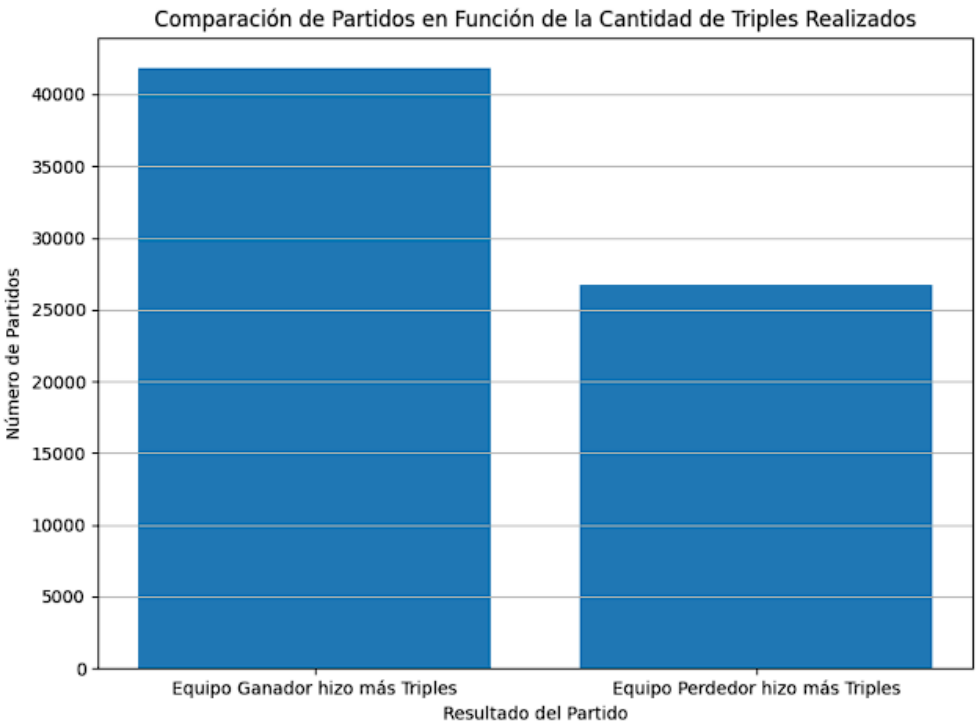
Al menos un 10% de las columnas han de ser categóricas: El 10% de las 34 columnas pertenecientes al data frame equivalen a 3 columnas redondeadas a entero (3,4 originalmente). Las columnas simuladas a ser establecidas como categóricas han sido: “Numot”, “Daynum” y “Wloc”.

```
[17] cdb1["Numot"] = pd.Categorical(cdb1.Numot)
     cdb1["Daynum"] = pd.Categorical(cdb1.Daynum)
     cdb1["Wloc"] = pd.Categorical(cdb1.Wloc)

[21] cdb1.columns

     Index(['Season', 'Daynum', 'Wteam', 'Wscore', 'Lteam', 'Lscore', 'Wloc',
            'Numot', 'Wfgm', 'Wfga', 'Wfgm3', 'Wfga3', 'Wftm', 'Wfta', 'Wor', 'Wdr',
            'Wast', 'Wto', 'Wstl', 'Wblk', 'Wpf', 'Lfgm', 'Lfga', 'Lfgm3', 'Lfga3',
            'Lftm', 'Lfta', 'Lor', 'Ldr', 'Last', 'Lto', 'Lstl', 'Lblk', 'Lpf'],
           dtype='object')
```

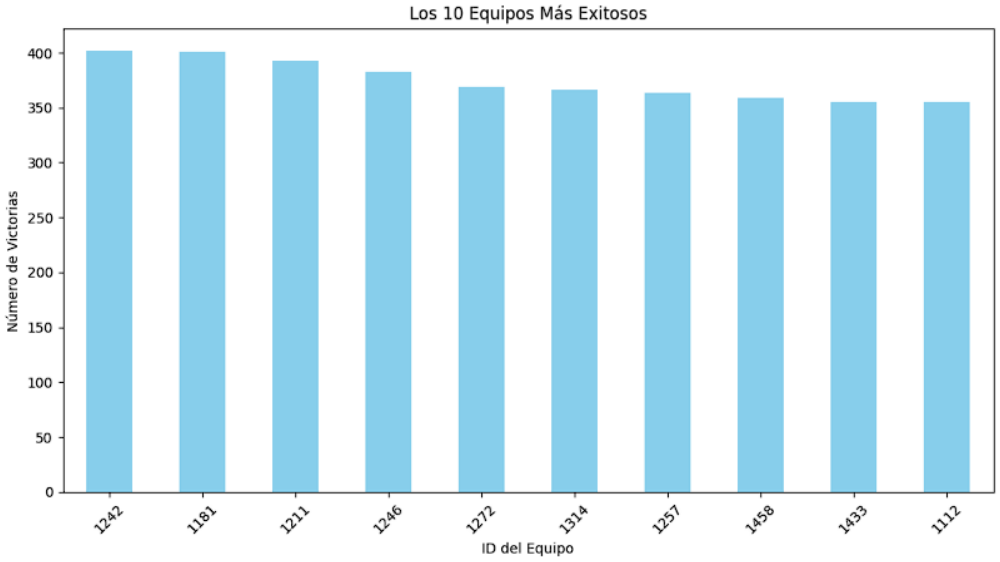
Exploración de datos. Inicialmente tomamos el conjunto de datos y revisamos las distintas variables que lo conforman, datos asociados a estas y la relación entre estos datos, todo esto para definir nuestro modelo de forma eficiente y acertada. Después de este proceso obtuvimos algunos gráficos en los que se reflejan estadísticas comparativas entre partidos como la cantidad de puntos anotados, los triples anotados, faltas, robos, etc. También se observan otras estadísticas relacionadas a los resultados de los encuentros.

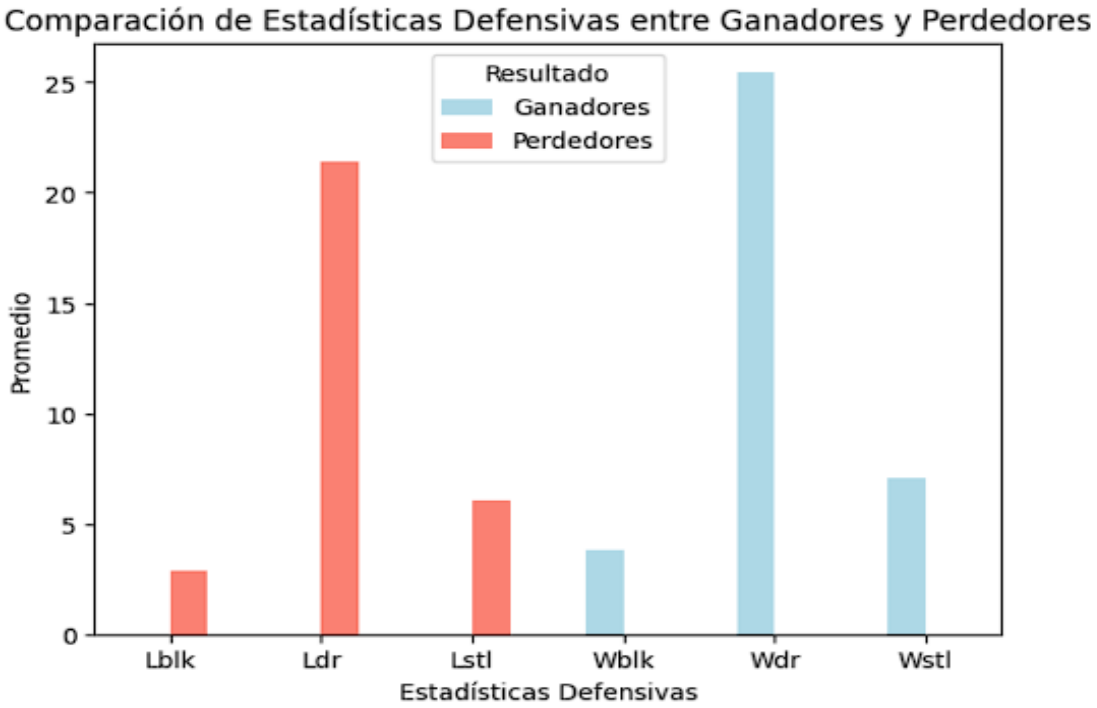


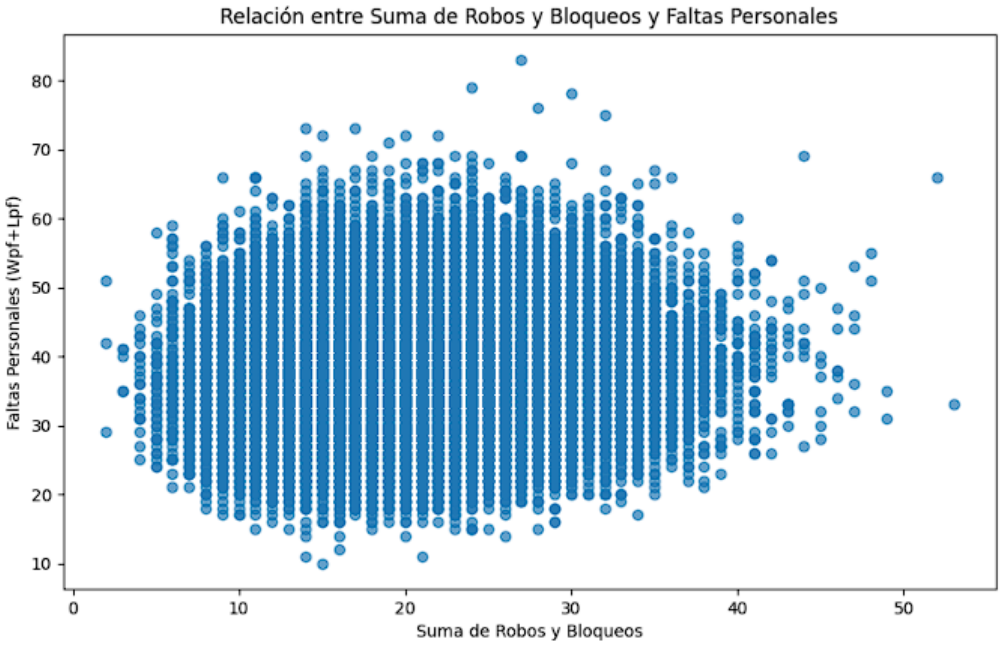
Observaciones: A medida que transcurren los años el baloncesto se vuelve un deporte más ofensivo, un deporte en el que se convierten cada vez más puntos , aunque existe una caída en los puntos en el año 2015 en comparación con el año anterior(2014).



Este gráfico muestra que los equipos ganadores en general convierten más triples que los equipos perdedores.



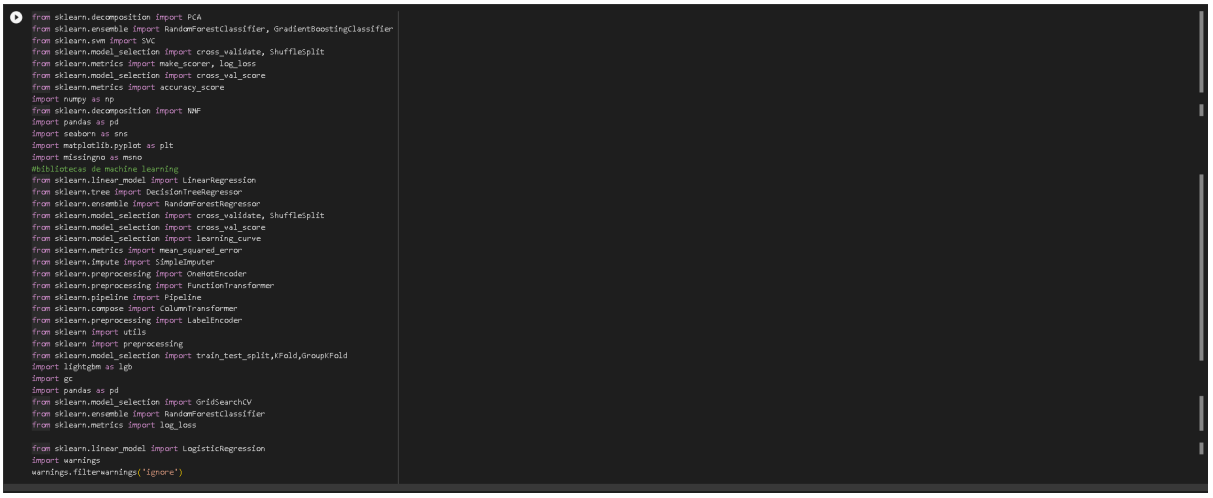




En este gráfico de dispersión, los bloqueos (wblk+lblq) más los robos (wstl+lstl) se representan en el eje x, y las faltas personales (wpf+lpf) en el eje y. Cada punto en el gráfico representa un partido ganado por un equipo. Si se observa una tendencia en la que equipos con más faltas personales también tienen más robos y bloqueos, podría sugerir que están dispuestos a asumir más riesgos en defensa para intentar obtener ventajas en el juego.

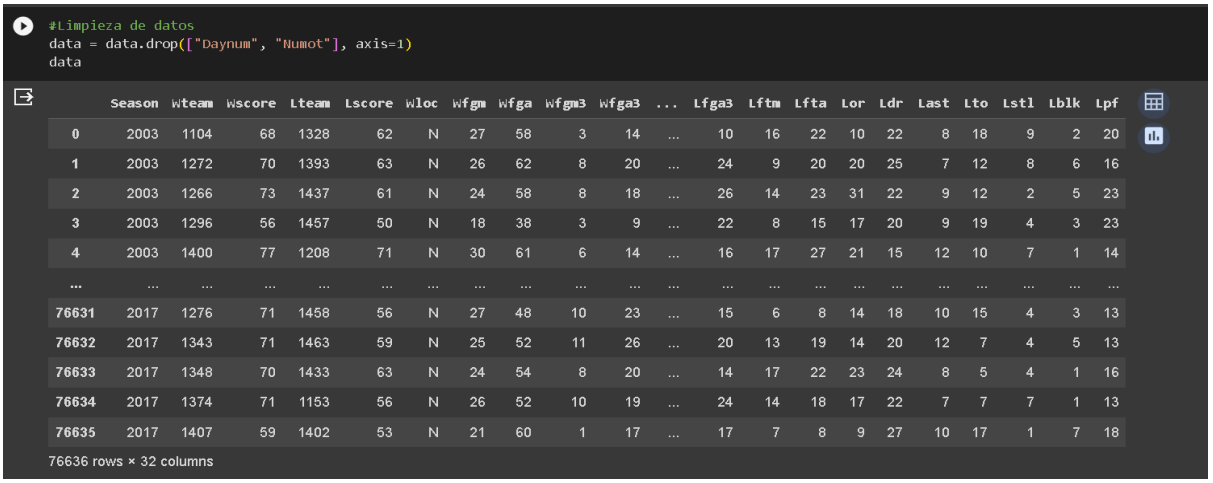
Limpieza de datos.

Se realizó una eliminación de las columnas más innecesarias del dataframe según lo indica la matriz de correlación realizada, esta matriz se interpreta de forma en que las celdas que se encuentren más cercanas a 1, son más propensas a replicar datos y por lo tanto se elimina una de las dos columnas del dataset, también se eliminan las columnas innecesarias “Daynum” y “Numot” las cuales contienen el día en que se jugó el partido y Numot el cual indica el número de períodos de tiempo extra en el juego. Además se hizo la conversión de la variable Wloc de categórica a numerica con un mapeo one-hot donde: ‘H’ es equipo local representado ahora con 0, ‘A’ es equipo visitante representado con 1, y ‘N’ es cancha neutral representada con 2.



```
from sklearn.decomposition import PCA
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.model_selection import cross_validate, ShuffleSplit
from sklearn.metrics import make_scorer, log_loss
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score
import numpy as np
from sklearn.decomposition import NMF
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import missingno as msno
#bibliotecas de machine learning
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import cross_validate, ShuffleSplit
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import learning_curve
from sklearn.metrics import mean_squared_error
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import FunctionTransformer
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import LabelEncoder
from sklearn import utils
from sklearn import preprocessing
from sklearn.model_selection import train_test_split,KFold,GroupKFold
import lightgbm as lgb
import gc
import pandas as pd
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import log_loss

from sklearn.linear_model import LogisticRegression
import warnings
warnings.filterwarnings('ignore')
```

```
[2] data=pd.read_csv('https://raw.githubusercontent.com/andresramirez31/datosModelos/main/RegularSeasonDetailedResults.csv')
    Teams=pd.read_csv('https://raw.githubusercontent.com/andresramirez31/datosModelos/main/Teams.csv')
```



```
#Limpieza de datos
data = data.drop(["Daynum", "Numot"], axis=1)
data
```

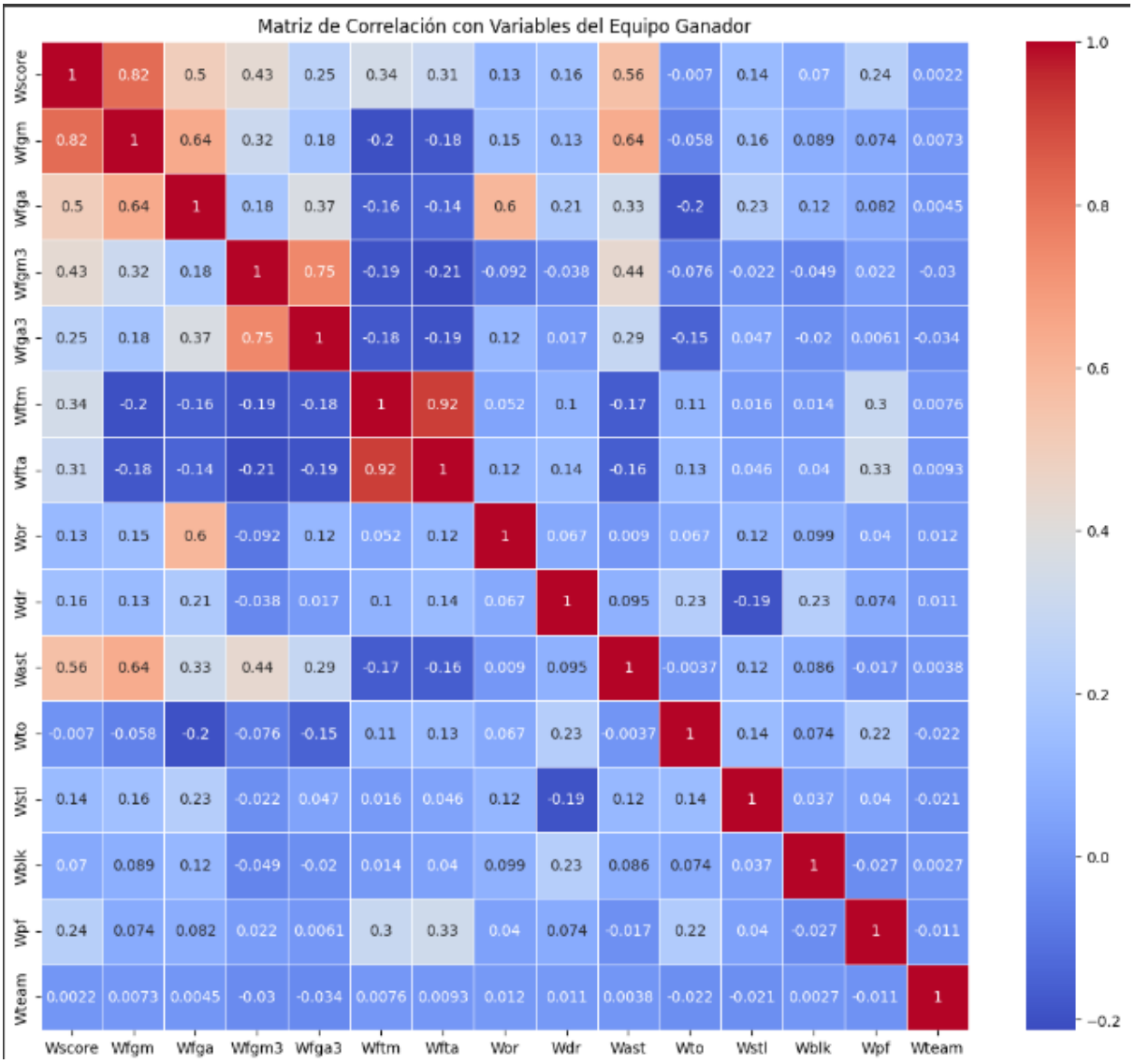
	Season	Wteam	Wscore	Lteam	Lscore	Wloc	Wfgm	Wfga	Wfgm3	Wfga3	...	Lfga3	Lftm	Lfta	Lor	Ldr	Last	Lto	Lstl	Lblk	Lpf
0	2003	1104	68	1328	62	N	27	58	3	14	...	10	16	22	10	22	8	18	9	2	20
1	2003	1272	70	1393	63	N	26	62	8	20	...	24	9	20	20	25	7	12	8	6	16
2	2003	1266	73	1437	61	N	24	58	8	18	...	26	14	23	31	22	9	12	2	5	23
3	2003	1296	56	1457	50	N	18	38	3	9	...	22	8	15	17	20	9	19	4	3	23
4	2003	1400	77	1208	71	N	30	61	6	14	...	16	17	27	21	15	12	10	7	1	14
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
76631	2017	1276	71	1458	56	N	27	48	10	23	...	15	6	8	14	18	10	15	4	3	13
76632	2017	1343	71	1463	59	N	25	52	11	26	...	20	13	19	14	20	12	7	4	5	13
76633	2017	1348	70	1433	63	N	24	54	8	20	...	14	17	22	23	24	8	5	4	1	16
76634	2017	1374	71	1153	56	N	26	52	10	19	...	24	14	18	17	22	7	7	7	1	13
76635	2017	1407	59	1402	53	N	21	60	1	17	...	17	7	8	9	27	10	17	1	7	18

76636 rows × 32 columns

```
[4] Wloc_mapping = {'H': 0, 'A': 1, 'N': 2}
    data['Wloc'] = data['Wloc'].map(Wloc_mapping)
```

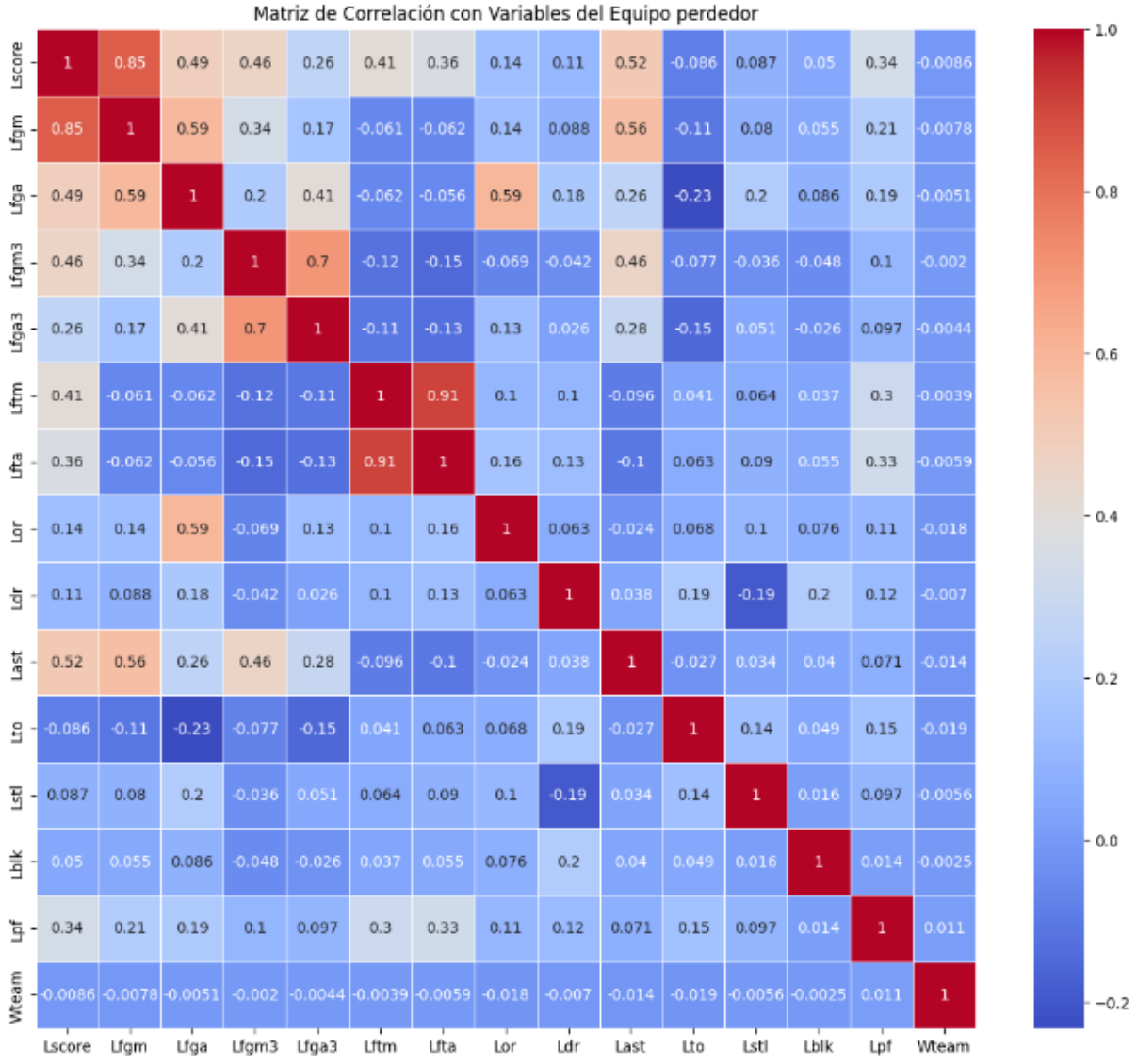
```
df=data


# Realiza el análisis de correlación y elimina las variables con menos correlación según sea necesario
# Calcular la matriz de correlación
columnas_numericas_equipo_ganador = ['Wscore', 'Wfgm', 'Wfga', 'Wfgm3', 'Wfga3', 'Wftm', 'Wfta', 'Wor', 'Wdr', 'Wast', 'Wto', 'Wstl', 'Wblk', 'Wpf']

# Crea un nuevo DataFrame solo con estas columnas
df_equipo_ganador_numericas = df[columnas_numericas_equipo_ganador]

# Añade la variable objetivo 'Wteam' al DataFrame
df_equipo_ganador_numericas['Wteam'] = df['Wteam']

# Calcula la matriz de correlación
correlacion_equipo_ganador = df_equipo_ganador_numericas.corr()

# Visualiza la matriz de correlación como un mapa de calor
plt.figure(figsize=(14, 12))
sns.heatmap(correlacion_equipo_ganador, annot=True, cmap='coolwarm', linewidths=.5)
plt.title("Matriz de Correlación con Variables del Equipo Ganador")
plt.show()
```

Gráfico que representa la matriz de correlación, donde se toman todas las variables relacionadas al equipo ganador.



```
df=data

columnas_numericas_equipo_perdedor = ['Lscore', 'Lfgm', 'Lfga', 'Lfgm3', 'Lfga3', 'Lftm', 'Lfta', 'Lor', 'Ldr', 'Last', 'Lto', 'Lstl', 'Lblk', 'Lpf']

# Crea un nuevo DataFrame solo con estas columnas
df_equipo_perdedor_numericas = df[columnas_numericas_equipo_perdedor]

# Añade la variable objetivo 'Wteam' al DataFrame
df_equipo_perdedor_numericas['Wteam'] = df['Wteam']

# Calcula la matriz de correlación
correlacion_equipo_perdedor = df_equipo_perdedor_numericas.corr()

# Visualiza la matriz de correlación como un mapa de calor
plt.figure(figsize=(14, 12))
sns.heatmap(correlacion_equipo_perdedor, annot=True, cmap='coolwarm', linewidths=.5)
plt.title("Matriz de Correlación con Variables del Equipo perdedor")
plt.show()
```

Gráfico que representa la matriz de correlación, donde se toman todas las variables relacionadas al equipo perdedor.



Particionamiento del dataset: Se asignaron las particiones Train y test correspondientes del dataset, usando una distribución porcentual de 70% entrenamiento (train) y 30% prueba (test) para la calibración y prueba de los modelos a utilizar.

```
[7] #eliminamos las variables con alta correlacion
    data = data.drop(['Lfgm', 'Wfgm'], axis=1)
```

```
df=data

# Supongamos que 'df' es tu DataFrame con los datos

# Selecciona las columnas que serán caracteristicas (X) y la variable objetivo (y)
columnas_caracteristicas = df.drop(['Wteam'], axis=1)
variable_objetivo = df['Wteam']
test_size=0.3
# Divide los datos en conjuntos de entrenamiento y prueba (80% entrenamiento, 20% prueba)
X_train, X_test, y_train, y_test = train_test_split(columnas_caracteristicas, variable_objetivo, test_size=test_size)

# Verifica las dimensiones de los conjuntos de entrenamiento y prueba
print("Dimensiones del conjunto de entrenamiento:", X_train.shape, y_train.shape)
print("Dimensiones del conjunto de prueba:", X_test.shape, y_test.shape)
```

```
Dimensiones del conjunto de entrenamiento: (53645, 29) (53645,)
Dimensiones del conjunto de prueba: (22991, 29) (22991,)
```

Modelos e iteraciones:

- ***Modelos Supervisados:***

Para desarrollar el análisis se plantean inicialmente tres modelos: RandomForestClassifier, LogisticRegression y Perceptron, luego se realizó un accuracy_score entre la predicción de cada modelo supervisado para la variable objetivo Wteam con respecto a los datos de prueba de la variable objetivo en el dataframe, lo cual nos brinda los siguientes resultados expuestos en la imagen posterior.

```
estimator1 = RandomForestClassifier(n_estimators=1, random_state=42)
estimator2 = LogisticRegression(random_state=42)
estimator3 = Perceptron(tol=1e-3, random_state=42)
estimators = [estimator1, estimator2, estimator3]
# Itera sobre los estimadores
for estimator in estimators:

    estimator.fit(X_train, y_train)


    y_pred = estimator.predict(X_test)


    accuracy_result = accuracy_score(y_test, y_pred)


    print(f"Accuracy para {estimator.__class__.__name__}: {accuracy_result}")
```

```
Accuracy para RandomForestClassifier: 0.009525466486886174
Accuracy para LogisticRegression: 0.014657909616806577
Accuracy para Perceptron: 0.003262146057152799
```

Definición de Hiperparametros:

- **RandomForest:** Teniendo en cuenta el análisis exhaustivo desarrollado con los hiperparametros de n_estimators, max_depth, min_samples_split y min_samples_leaf para el modelo RandomForest, se obtuvieron los siguientes resultados como los mejores hiperparametros: max_depth = 10, min_samples_leaf = 1, min_samples_split = 10, n_estimators= 15.

```
# Define el modelo
rf_model = RandomForestClassifier(random_state=42)

# Define los hiperparámetros a ajustar
param_grid_rf = {
    'n_estimators': [5, 10, 15],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Realiza la búsqueda en cuadrícula
grid_search_rf = GridSearchCV(estimator=rf_model, param_grid=param_grid_rf, cv=5, scoring='accuracy')
grid_search_rf.fit(X_train, y_train)

# Imprime los mejores hiperparámetros y la precisión asociada
print("Best Hyperparameters for Random Forest:")
print(grid_search_rf.best_params_)
print("Mejor Accuracy:", grid_search_rf.best_score_)
```

```
Best Hyperparameters for Random Forest:
{'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split': 10, 'n_estimators': 15}
Mejor Accuracy: 0.022481125920402648
```

```
[16] #implementacion mejores hiperparametros
     RanFor = RandomForestClassifier(max_depth = 10, min_samples_leaf = 1, min_samples_split = 10, n_estimators = 15)
     RanFor.fit(X_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(max_depth=10, min_samples_split=10, n_estimators=15)
```

- **Perceptron:** Teniendo en cuenta el análisis exhaustivo desarrollado con los hiperparametros de alpha y penalty para el modelo Perceptron, se obtuvieron los siguientes resultados como los mejores hiperparametros: alpha=0.001 y penalty= l1.

```
from sklearn.linear_model import Perceptron
P = Perceptron(tol=1e-3, random_state=42)

# Define los hiperparámetros a ajustar
param_grid_p = {
    'alpha': [0.001, 0.01, 0.1, 1, 10],
    'penalty': ['l1', 'l2']
}

# Realiza la búsqueda en cuadrícula
grid_search_p = GridSearchCV(estimator=P, param_grid=param_grid_p, cv=5, scoring='accuracy')
grid_search_p.fit(X_train, y_train)

# Imprime los mejores hiperparámetros y la precisión asociada
print("Mejores hiperparametros para Perceptron:")
print(grid_search_p.best_params_)
print("Mejor Accuracy:", grid_search_p.best_score_)
```

```
Mejores hiperparametros para Perceptron:
{'alpha': 0.001, 'penalty': 'l1'}
Mejor Accuracy: 0.0041569577779849
```

```
[12] #implementacion mejores hiperparametros
     paramP = Perceptron(alpha= 0.001, penalty= 'l1')
     paramP.fit(X_train, y_train)
```

```
Perceptron
Perceptron(alpha=0.001, penalty='l1')
```

- **LogisticRegression:** Teniendo en cuenta el análisis exhaustivo desarrollado con los hiperparametros de C y penalty para el modelo LogisticRegression, se obtuvieron los siguientes resultados como los mejores hiperparametros: C=0.01 y penalty= l2.

```
lr_model = LogisticRegression(random_state=42)

# Define los hiperparámetros a ajustar
param_grid_lr = {
    'C': [0.001, 0.01, 0.1, 1, 10],
    'penalty': ['l1', 'l2']
}

# Realiza la búsqueda en cuadrícula
grid_search_lr = GridSearchCV(estimator=lr_model, param_grid=param_grid_lr, cv=5, scoring='accuracy')
grid_search_lr.fit(X_train, y_train)

# Imprime los mejores hiperparámetros y la precisión asociada
print("Mejores hiperparametros para Logistic Regression:")
print(grid_search_lr.best_params_)
print("Mejor Accuracy:", grid_search_lr.best_score_)
```

```
Mejores hiperparametros para Logistic Regression:
{'C': 0.01, 'penalty': 'l2'}
Mejor Accuracy: 0.01703793456985739 5
```

```
[14] #implementacion mejores hiperparametros
     Logis = LogisticRegression(C= 0.01, penalty= 'l2')
     Logis.fit(X_train, y_train)
```

```
LogisticRegression(C=0.01)
```

- ***Modelos no supervisados:***

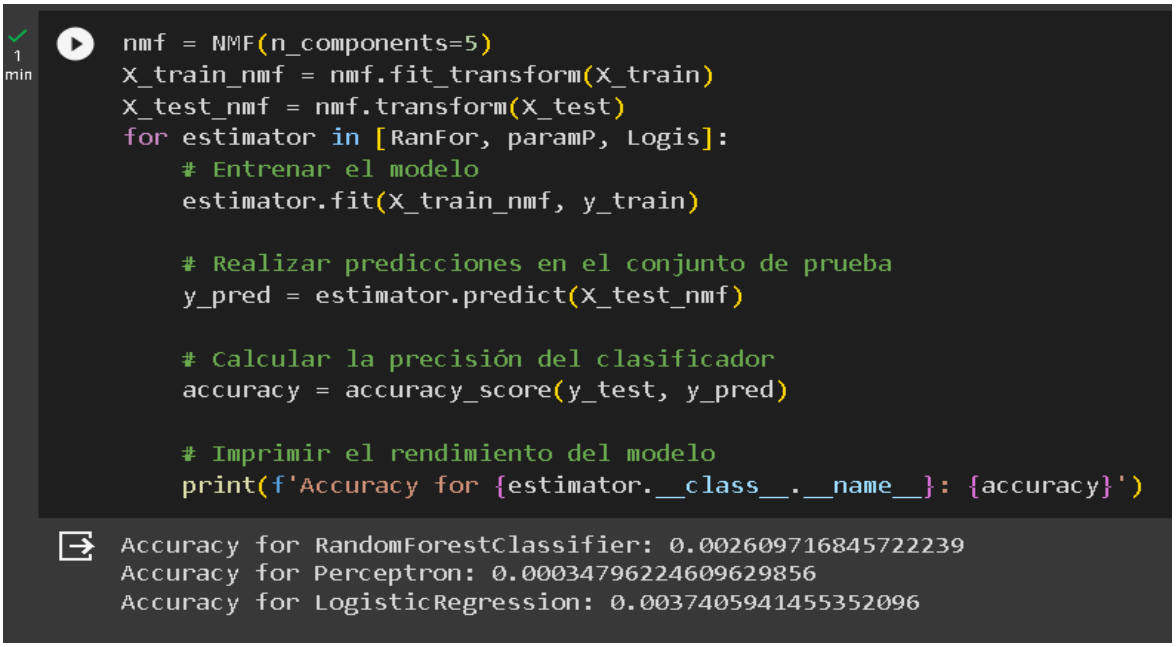
También se plantearon dos modelos no supervisados.

- **PCA:** Para implementar el PCA, se realizó un código mediante el cual se exploraron varias opciones de hiper parámetro de este algoritmo. Tomando valores para su hiperparametros n components de 3 y 5, apenas escogimos estos 2 por los largos tiempos de ejecución del código. por la ejecución se puede visualizar que el mejor PCA es n components=5 ya que los puntajes de accuracy son mayores

```
# Aplica PCA para reducir la dimensionalidad
n_components = [3, 5]  # Elige el número de componentes principales que deseas retener
for i in n_components:
  pca = PCA(n_components=i)
  X_train_pca = pca.fit_transform(X_train)
  X_test_pca = pca.transform(X_test)
  # Entrenar los modelos con los datos de entrenamiento después de PCA
  RanFor.fit(X_train_pca, y_train)
  paramP.fit(X_train_pca, y_train)
  Logis.fit(X_train_pca, y_train)
  # Realizar predicciones en el conjunto de prueba después de PCA
  rf_predictions = RanFor.predict(X_test_pca)
  paramP_predictions = paramP.predict(X_test_pca)
  lr_predictions = Logis.predict(X_test_pca)
  # Evaluar el rendimiento después de PCA utilizando accuracy
  rf_accuracy = accuracy_score(y_test, rf_predictions)
  paramP_accuracy = accuracy_score(y_test, paramP_predictions)
  lr_accuracy = accuracy_score(y_test, lr_predictions)
  print("Random Forest Accuracy despues de PCA:", rf_accuracy)
  print("Perceptron Accuracy despues de PCA:", paramP_accuracy)
  print("Logistic Regression Accuracy despues de PCA:", lr_accuracy)
```

```
Random Forest Accuracy despues de PCA: 0.017093645339480667
Perceptron Accuracy despues de PCA: 0.002131268757339829
Logistic Regression Accuracy despues de PCA: 0.008960027836979688
Random Forest Accuracy despues de PCA: 0.023269975207689964
Perceptron Accuracy despues de PCA: 0.003871079987821321
Logistic Regression Accuracy despues de PCA: 0.013396546474707494
```

- **NMF:** Para el caso de NMF hicimos la prueba con n componentes=5 y con todos los estimadores, se puede visualizar en la ejecución que el mejor sería Logistic Regression, ya que nos entre el accuracy más alto



```
nmf = NMF(n_components=5)
X_train_nmf = nmf.fit_transform(X_train)
X_test_nmf = nmf.transform(X_test)
for estimator in [RanFor, paramP, Logis]:
    # Entrenar el modelo
    estimator.fit(X_train_nmf, y_train)

    # Realizar predicciones en el conjunto de prueba
    y_pred = estimator.predict(X_test_nmf)

    # Calcular la precisión del clasificador
    accuracy = accuracy_score(y_test, y_pred)

    # Imprimir el rendimiento del modelo
    print(f'Accuracy for {estimator.__class__.__name__}: {accuracy}')
```

```
Accuracy for RandomForestClassifier: 0.002609716845722239
Accuracy for Perceptron: 0.00034796224609629856
Accuracy for LogisticRegression: 0.0037405941455352096
```

Curvas de aprendizaje:

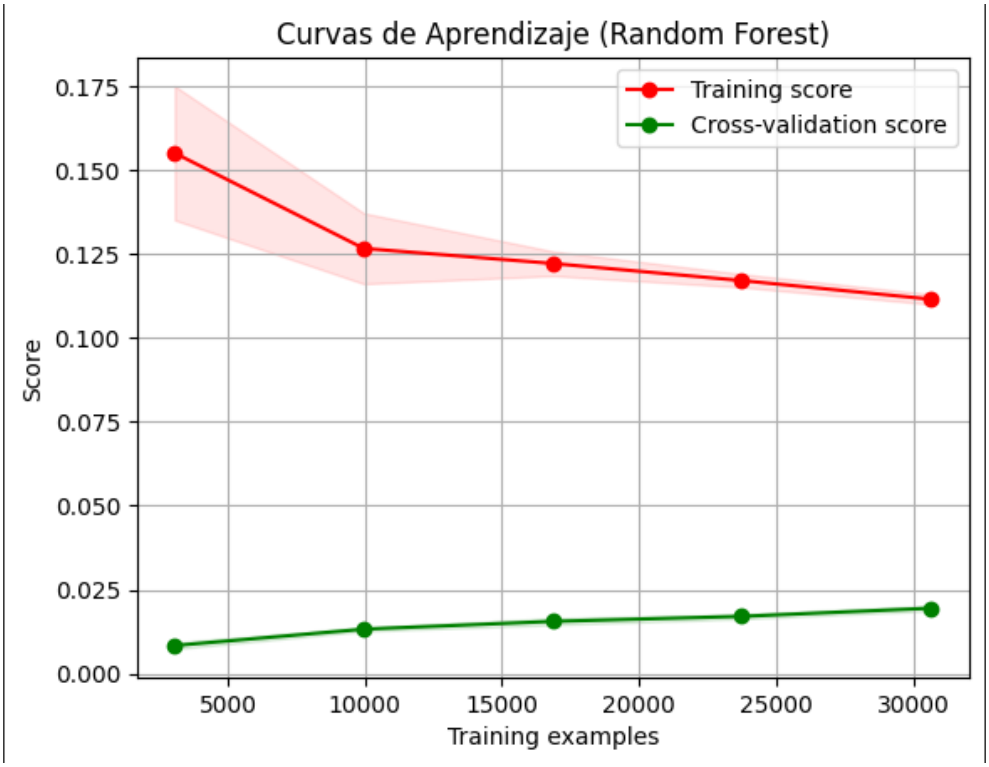
Se realizaron los análisis y gráficas de las curvas de aprendizaje respectivas para tanto los modelos supervisados utilizados (RandomForest, Perceptron, LogisticRegression) como los no supervisados(PCA y NMF), con el objetivo de ayudarnos a evaluar el rendimiento de cada modelo en el transcurso del aumento del tamaño del dataset.

Modelos supervisados:

- **RandomForest:**

El análisis con RandomForest demuestra una curva de aprendizaje deficiente donde los resultados de entrenamiento y los de validación se encuentran notablemente aislados el uno del otro, además de demostrar una tendencia a la disminución del ratio de entrenamiento, lo cual nos dice que el modelo tiene una capacidad de aprendizaje deficiente, de esta gráfica se puede inferir que los resultados denotan una curva de aprendizaje insuficiente, y por ende, no se podrá considerar el rendimiento de este modelo como satisfactorio para nuestra solución del problema.

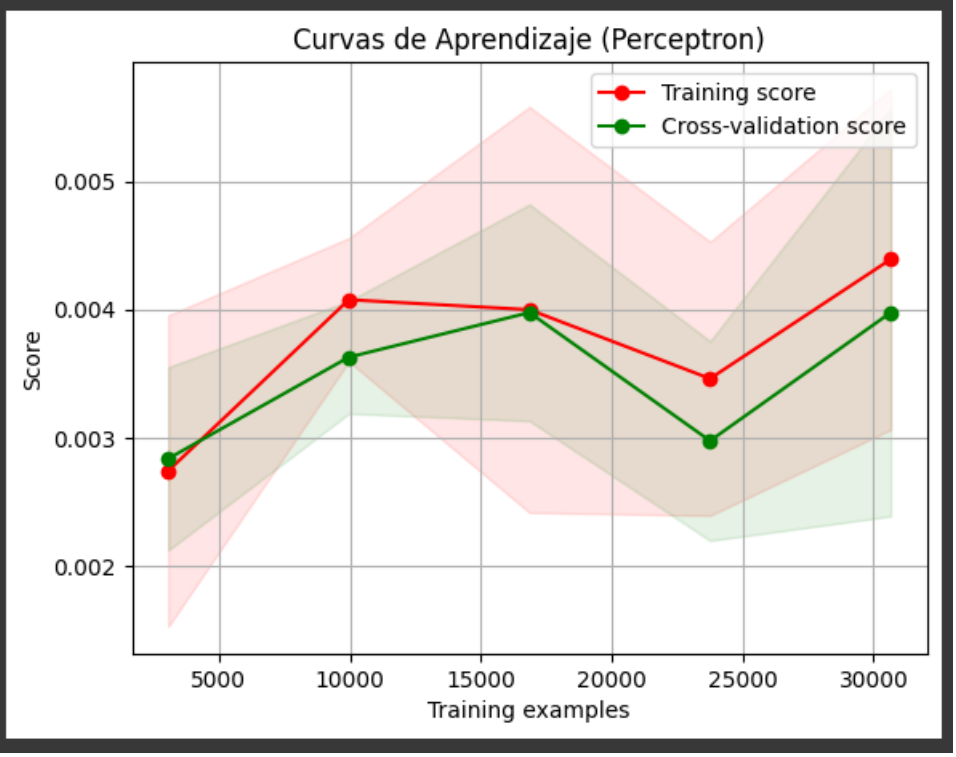
```
[19] model = RanFor
     test_size = 0.3
     val_size  = test_size/(1-test_size)
     # Función para crear curvas de aprendizaje
     def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None, n_jobs=None, train_sizes=np.linspace(.1, 1.0, 5)):
         plt.figure()
         plt.title(title)
         if ylim is not None:
             plt.ylim(*ylim)
         plt.xlabel("Training examples")
         plt.ylabel("Score")

         train_sizes, train_scores, test_scores = learning_curve(
             estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes)

         train_scores_mean = np.mean(train_scores, axis=1)
         train_scores_std = np.std(train_scores, axis=1)
         test_scores_mean = np.mean(test_scores, axis=1)
         test_scores_std = np.std(test_scores, axis=1)

         plt.grid()

         plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                          train_scores_mean + train_scores_std, alpha=0.1,
                          color="r")
         plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                          test_scores_mean + test_scores_std, alpha=0.1, color="g")

         plt.plot(train_sizes, train_scores_mean, 'o-', color="r", label="Training score")
         plt.plot(train_sizes, test_scores_mean, 'o-', color="g", label="Cross-validation score")

         plt.legend(loc="best")
         return plt

     # Crear curvas de aprendizaje
     title = "Curvas de Aprendizaje (Random Forest)"
     cv = ShuffleSplit(n_splits=5, test_size=val_size)
     plot_learning_curve(model, title, X_train, y_train, cv=cv, n_jobs=4)

     plt.show()
```



- **Perceptron:**

El análisis con Perceptron demuestra una curva de aprendizaje interesante donde los resultados de entrenamiento y los de validación no se encuentran completamente aislados el uno del otro, pero tampoco demuestran una tendencia a la disminución, lo cual nos dice que el modelo tiene una capacidad de aprendizaje adecuada, de esta gráfica se puede inferir que los resultados denotan una curva de aprendizaje de buen ajuste.

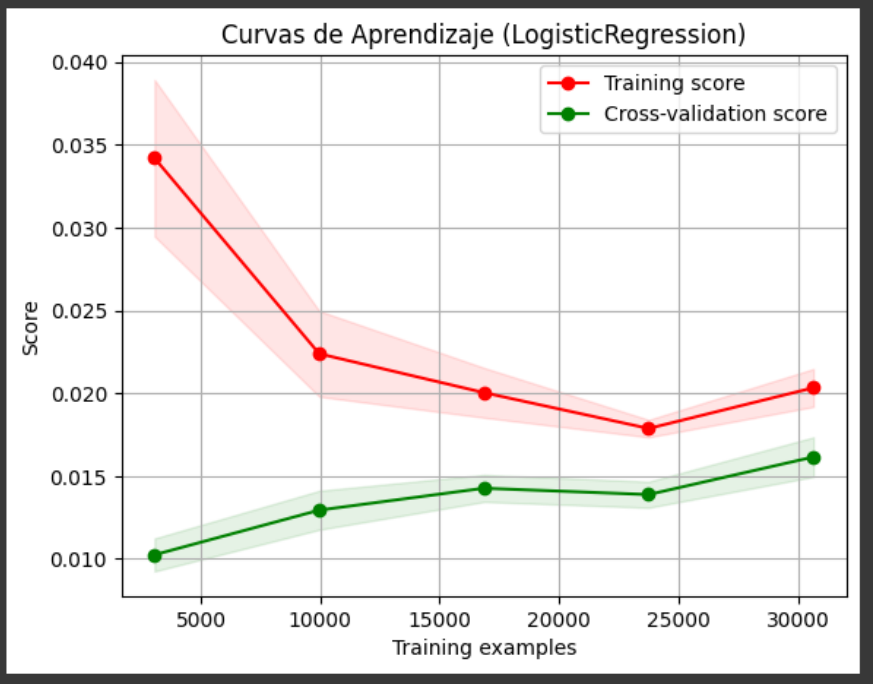
```
[20] model = paramP

     # Crear curvas de aprendizaje
     title = "Curvas de Aprendizaje (Perceptron)"
     cv = ShuffleSplit(n_splits=5, test_size=val_size)
     plot_learning_curve(model, title, X_train, y_train, cv=cv, n_jobs=4)

     plt.show()
```



- **LogisticRegression:**

El análisis con LogisticRegression demuestra una curva de aprendizaje potencial que donde los resultados de entrenamiento y los de validación demuestran un progresivo acercamiento el uno del otro, y además , podemos evidenciar que al final la curva de entrenamiento muestra una inclinación al aumento, lo cual nos dice que el modelo tiene también la capacidad de un aprendizaje eficiente. Por ende, podemos decir que los resultados denotan una curva de aprendizaje potencial y se podrá tener en cuenta el rendimiento de este modelo para nuestra solución del problema.

```
[21] model = Logis

     # Crear curvas de aprendizaje
     title = "Curvas de Aprendizaje (LogisticRegression)"
     cv = ShuffleSplit(n_splits=5, test_size=val_size)
     plot_learning_curve(model, title, X_train, y_train, cv=cv, n_jobs=4)

     plt.show()
```

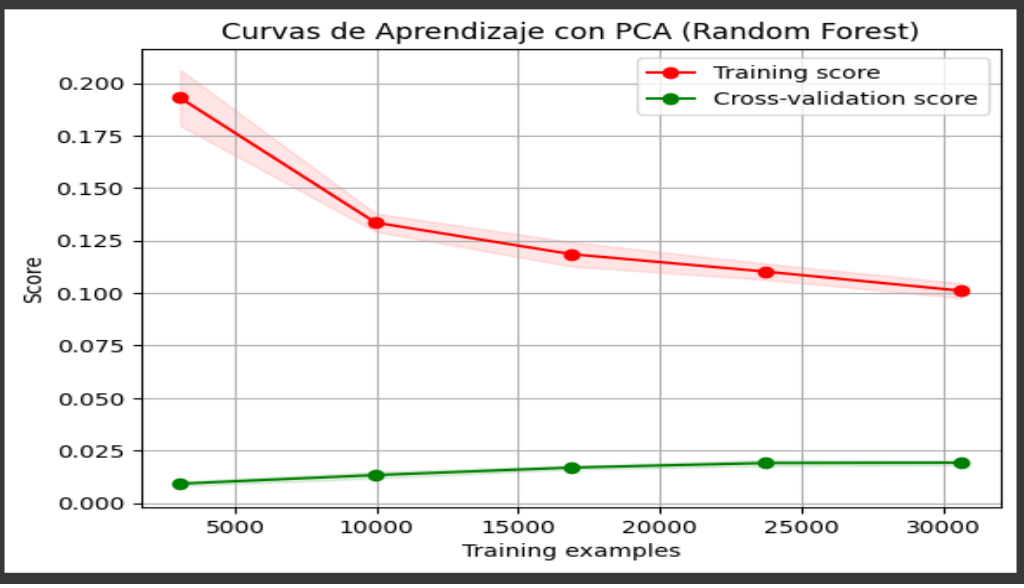


Modelos no supervisados:

- **PCA:**

El análisis con PCA demuestra una curva de aprendizaje deficiente donde los resultados de entrenamiento y los de validación tienen un moderado aislamiento el uno del otro, se evidencia una tendencia a la disminución del ratio de entrenamiento. Esto nos dice que el modelo tiene una capacidad de aprendizaje deficiente. Por ende, podemos decir que los resultados denotan una curva de aprendizaje insuficiente y no se podrá considerar el rendimiento de este modelo como satisfactorio para nuestra solución del problema.

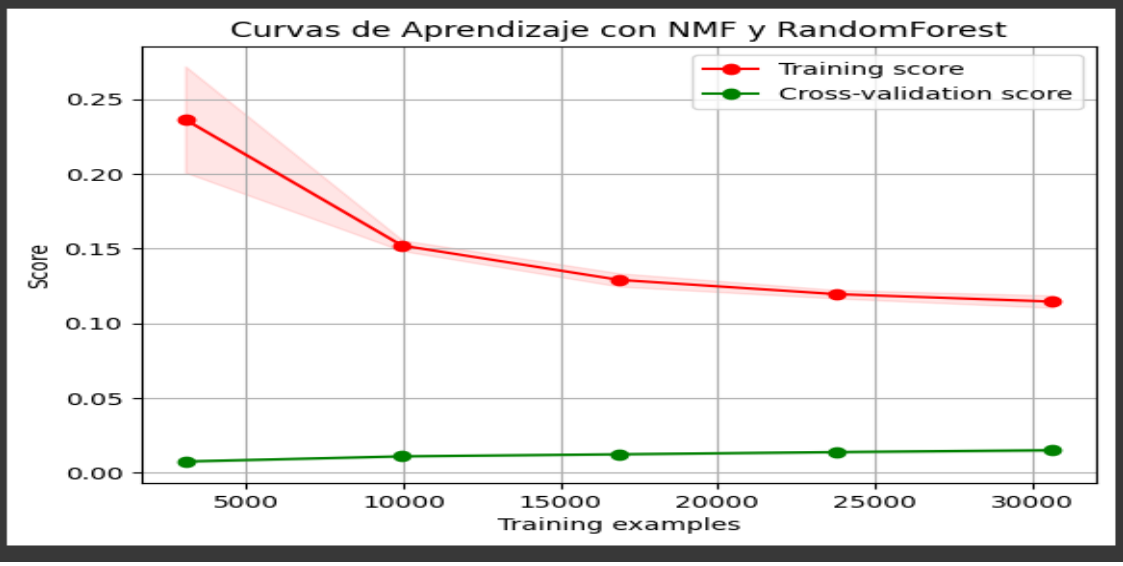
```
title = "Curvas de Aprendizaje con PCA (Random Forest)"
cv = ShuffleSplit(n_splits=5, test_size=val_size)
plot_learning_curve(RanFor, title, X_train_pca, y_train, cv=cv, n_jobs=4)

plt.show()
```



- **NMF:**

El análisis con NMF demuestra una curva de aprendizaje deficiente donde los resultados de entrenamiento y los de validación tienen un moderado aislamiento el uno del otro, se evidencia una tendencia a la disminución del ratio de entrenamiento. Esto nos dice que el modelo tiene una capacidad de aprendizaje deficiente. Por ende, podemos decir que los resultados denotan una curva de aprendizaje insuficiente y no se podrá considerar el rendimiento de este modelo como satisfactorio para nuestra solución del problema.

```
# Aplicar NMF a tus datos

# Crear curvas de aprendizaje con NMF
title = "Curvas de Aprendizaje con NMF y RandomForest"
plot_learning_curve(RanFor, title, X_train_nmf, y_train, cv=cv, n_jobs=4)

plt.show()
```



Retos y conclusiones:

El proyecto de predicción de resultados del torneo de baloncesto universitario masculino en EE. UU. en 2017 presentó una serie de desafíos importantes.

La limpieza y preparación de datos fue un punto crítico. Trabajar con un conjunto extenso y complejo, con múltiples temporadas y muchas columnas, demandó un esfuerzo considerable en el manejo de valores faltantes y la transformación de datos categóricos.

Seleccionar las características más relevantes para predecir los resultados del torneo fue otro desafío. La utilización de herramientas como la matriz de correlación fue fundamental para entender las relaciones entre variables y decidir qué columnas mantener o eliminar para mejorar los modelos.

La elección y ajuste de modelos fue un proceso arduo. Probar varios modelos supervisados y no supervisados, ajustar sus hiperparámetros y evaluar su desempeño a través de métricas como la precisión y las curvas de aprendizaje fue clave para encontrar los modelos más adecuados para este problema específico.

El manejo del desbalance de datos también representó un reto. En problemas de predicción deportiva, la proporción de victorias y derrotas puede ser desigual, requiriendo técnicas específicas para evitar sesgos en los modelos.

Finalmente, interpretar los resultados y aplicarlos a la predicción precisa del torneo fue fundamental. Comunicar de manera efectiva estos hallazgos para su implementación práctica fue crucial para el éxito del proyecto.

En conclusión, este proyecto implicó un análisis exhaustivo que partió desde la exploración detallada de los datos hasta la implementación y evaluación de modelos. Los resultados destacaron la efectividad de los modelos LogisticRegression y Perceptron para predecir con precisión los resultados del torneo, revelando descubrimientos interesantes sobre la evolución del juego a lo largo de los años y las estrategias de los equipos ganadores

Bibliografía