



CASO PRÁCTICO 7

MÓDULO PYTHON

Juan Manuel Mendoza Ramos

Contenido:

1. **Instrucciones.**
2. **Descripción del problema o pregunta:**
 - a. ¿Qué intentas predecir, analizar o investigar con los datos seleccionados?
 - b. Describe el contexto del conjunto de datos y por qué es relevante para el análisis.
3. **Análisis Exploratorio de Datos (EDA):**
 - a. Realiza un análisis inicial para comprender la estructura de los datos. Identifica valores faltantes, distribuciones y patrones.
 - b. Crea **visualizaciones** (gráficos de barras, histogramas, gráficos de dispersión) que ayuden a visualizar los datos y su distribución.
4. **Preparación de los Datos:**
 - a. Limpiar los datos: Manejar valores faltantes, duplicados o incorrectos.
 - b. Normalizar o escalar los datos si es necesario.
 - c. Convertir variables categóricas en variables numéricas (si aplica).
5. **Modelado Predictivo:**
 - a. Construir un modelo predictivo utilizando **Scikit-learn**. Ejemplos de modelos que pueden utilizar:
 - i. **Regresión lineal** para predicción de valores continuos.
 - ii. **Clasificación** (Regresión logística, KNN) si la variable objetivo es categórica.
 - b. Explicar brevemente la elección del modelo.
6. **Evaluación del Modelo:**
 - a. Utilizar métricas como **Precisión, Recall, F1 Score** (si es un problema de clasificación) o **Mean Squared Error (MSE)** (si es un problema de regresión).
 - b. Mostrar gráficos como la **Curva ROC** (para clasificación) o **gráfico de errores residuales** (para regresión).
7. **Visualización de Resultados:**
 - a. Generar visualizaciones que resuman los resultados del análisis predictivo. Estas pueden incluir gráficos de dispersión con las predicciones, gráficos de barras, histogramas, etc.
8. **Conclusiones y Recomendaciones:**
 - a. Responder a la pregunta inicial o problema planteado con base en los resultados obtenidos.
 - b. Proporcionar posibles mejoras o recomendaciones para trabajos futuros.
9. **Despliegue del Modelo:**
 - a. (Opcional) Explica cómo podrías desplegar el modelo en un entorno productivo, como una API o una aplicación web.

1. Instrucciones.

Aplicación de Ciencia y Analítica de Datos con Python

Título del Proyecto: Análisis Predictivo de un Conjunto de Datos de Interés

Los participantes del curso llevarán a cabo un **proyecto de ciencia y analítica de datos** utilizando **Python**. El objetivo es aplicar los conceptos teóricos y prácticos aprendidos durante el curso, desde la manipulación de datos con **NumPy** y **Pandas**, hasta la creación de visualizaciones, el desarrollo de modelos predictivos básicos, y su posterior evaluación y despliegue.

Objetivo del Proyecto:

El **proyecto final** tiene como objetivo que los participantes seleccionen un conjunto de datos **propios o de interés**, realicen un análisis exploratorio, construyan un modelo predictivo y desarrollen visualizaciones para comunicar los resultados. Los datos pueden ser de áreas como:

- **Ventas o Marketing** (ventas por mes, análisis de clientes).
- **Salud** (análisis de pacientes, diagnósticos, tratamientos).
- **Deportes** (rendimiento de jugadores, resultados de partidos).
- **Educación** (desempeño académico, tasa de asistencia).

Los participantes deberán crear una solución analítica que responda a una pregunta específica o problema utilizando las herramientas de análisis utilizadas, de manera individual o en equipo de máximo 4 integrantes, pero **TODOS** deben entregar.

Entregables del Proyecto:

Cada equipo o participante debe entregar los siguientes elementos:

1. **Descripción del problema o pregunta:**
 - ¿Qué intentas predecir, analizar o investigar con los datos seleccionados?
 - Describe el contexto del conjunto de datos y por qué es relevante para el análisis.
2. **Análisis Exploratorio de Datos (EDA):**
 - Realiza un análisis inicial para comprender la estructura de los datos. Identifica valores faltantes, distribuciones y patrones.
 - Crea **visualizaciones** (gráficos de barras, histogramas, gráficos de dispersión) que ayuden a visualizar los datos y su distribución.

3. Preparación de los Datos:

- Limpiar los datos: Manejar valores faltantes, duplicados o incorrectos.
- Normalizar o escalar los datos si es necesario.
- Convertir variables categóricas en variables numéricas (si aplica).

4. Modelado Predictivo:

- Construir un modelo predictivo utilizando **Scikit-learn**. Ejemplos de modelos que pueden utilizar:
 - **Regresión lineal** para predicción de valores continuos.
 - **Clasificación** (Regresión logística, KNN) si la variable objetivo es categórica.
- Explicar brevemente la elección del modelo.

5. Evaluación del Modelo:

- Utilizar métricas como **Precisión, Recall, F1 Score** (si es un problema de clasificación) o **Mean Squared Error (MSE)** (si es un problema de regresión).
- Mostrar gráficos como la **Curva ROC** (para clasificación) o **gráfico de errores residuales** (para regresión).

6. Visualización de Resultados:

- Generar visualizaciones que resuman los resultados del análisis predictivo. Estas pueden incluir gráficos de dispersión con las predicciones, gráficos de barras, histogramas, etc.

7. Conclusiones y Recomendaciones:

- Responder a la pregunta inicial o problema planteado con base en los resultados obtenidos.
- Proporcionar posibles mejoras o recomendaciones para trabajos futuros.

8. Despliegue del Modelo:

- (Opcional) Explica cómo podrías desplegar el modelo en un entorno productivo, como una API o una aplicación web.
-

Rúbrica de Evaluación (Total: 100 puntos)

Criterio	Puntos	Descripción de la Evidencia Esperada
1. Descripción del Problema	10	- El participante describe claramente la pregunta o problema que el análisis intenta resolver. - Explicación de la relevancia del conjunto de datos seleccionado. - Se presentan gráficos para visualizar la distribución de los datos.
2. Análisis Exploratorio de Datos	20	- Identificación de valores atípicos o faltantes. - Comentarios sobre las relaciones entre variables.
3. Preparación de los Datos	15	- Los datos se limpian adecuadamente, con un enfoque claro en la eliminación de valores faltantes y duplicados. - Variables categóricas convertidas correctamente (si es necesario). - Elección justificada del modelo (regresión, clasificación, etc.).
4. Modelado Predictivo	20	- Implementación del modelo de aprendizaje automático correctamente ajustado. - Explicación de las variables utilizadas.
5. Evaluación del Modelo	15	- Utilización de las métricas adecuadas para evaluar el rendimiento del modelo. - Explicación clara de los resultados obtenidos y su interpretación.
6. Visualización de Resultados	10	- Las visualizaciones son claras y bien explicadas. - Los gráficos respaldan las conclusiones y proporcionan un resumen visual del análisis.
7. Conclusiones y Recomendaciones	10	- El análisis responde a la pregunta inicial. - Se proporcionan recomendaciones o ideas para mejorar el análisis o seguir trabajando con el conjunto de datos.
8. Despliegue del Modelo (Opcional)	10 (Opc.)	- Descripción clara de cómo el modelo podría ser implementado en un entorno productivo, por ejemplo, en una API o aplicación.

Evidencias Esperadas

- **Código en Python** utilizando **Pandas, NumPy, Matplotlib/Seaborn y Scikit-learn** para el análisis y modelado predictivo.
- **Informe escrito** en el cual se detalle cada uno de los pasos del proyecto, las decisiones tomadas y las conclusiones alcanzadas.
- **Visualizaciones gráficas** que respalden el análisis y resumen los resultados.

Comentarios

Este proyecto permite a los participantes aplicar de forma integral lo aprendido durante el curso de **Ciencia y Analítica de Datos con Python**, trabajando con sus propios datos o seleccionando un conjunto de datos de su interés. Esto les brinda la oportunidad de obtener experiencia práctica en la manipulación de datos, construcción de modelos predictivos, evaluación de resultados y presentación de sus hallazgos de manera clara y concisa. Entrega tardía al día siguiente, misma hora.

2. Descripción del problema o pregunta.

CONTEXTO

“Análisis Predictivo de un Conjunto de Datos de jugadores de liga universitaria de Basketball”

Se dispone de un conjunto de datos que describen Nombre, Universidad de origen, las condiciones físicas, posición en la duela y logros obtenidos durante el último año. Con base a lo anterior, los directivos de la liga, desean aportar información a los equipos de ligas mayores para reforzar la ofensiva y conservar los jugadores que más aportan.

OBJETIVO

Determinar los jugadores que lograron más de 500 puntos durante la temporada y evaluar los resultados para el pronóstico.

3. Análisis Exploratorio de Datos (EDA):

Se preparó la base de datos xls y se convirtió a csv.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	id	player	height	weight	college	age	two_points	two_point_percentage	fts	ast	pts	mayor500	
2	0	Cliff Barker	188	83	University of Kentucky	29	102	37%	106	109	279	0	
3	1	Ralph Beard	178	79	University of Kentucky	22	340	36%	282	233	895	1	
4	2	Charlie Black	196	90	University of Kansas	28	226	28%	321	163	661	1	
5	3	Nelson Bobb	183	77	Temple University	25	80	32%	131	46	242	0	
6	4	Jake Bornheimer	196	90	Muhlenberg College	22	88	29%	117	40	254	0	
7	5	Vince Boryla	196	95	University of Denver	22	204	34%	267	95	612	1	
8	6	Don Bowen	193	95	Western Michigan University	24	208	37%	349	137	656	1	
9	7	Harry Boykoff	208	102	St. John's University	27	288	41%	262	149	779	1	
10	8	Joe Bradley	190	79	Oklahoma State University	21	36	27%	38	36	87	0	
11	9	Carl Braun	196	81	Colgate University	22	373	36%	374	247	1031	1	
12	10	Frankie Brian	185	81	Louisiana State University	26	368	32%	488	189	1138	1	
13	11	Price Brookfield	193	83	West Texas A&M University	29	11	48%	13	1	34	0	
14	12	Bob Brown	193	92	Miami University	26	276	36%	252	101	724	1	
15	13	Walt Budko	196	99	Columbia University	24	198	30%	263	146	595	1	
16	14	Tommy Byrnes	190	79	Seton Hall University	26	120	30%	124	88	327	0	
17	15	Bill Calhoun	190	81	City College of San Francisco	22	207	38%	203	115	560	1	
18	16	Don Carlson	183	77	University of Minnesota	30	99	34%	95	76	267	0	
19	17	Leroy Chollet	188	86	Canisius College	24	61	34%	56	37	157	0	
20	18	Bill Closs	196	88	Rice University	28	283	32%	259	160	752	1	
21	19	Jack Coleman	201	88	University of Louisville	25	250	38%	121	153	590	1	
22	20	Jack Cotton	201	90	University of Wyoming	25	97	29%	161	65	276	0	
23	21	Dillard Crocker	193	92	Western Michigan University	25	245	29%	317	85	723	1	
24	22	Chink Crossin	185	74	University of Pennsylvania	26	185	32%	101	148	449	0	
25	23	Fran Curran	183	79	University of Notre Dame	27	98	42%	241	71	395	0	
26	24	Bob Davies*	185	79	Seton Hall University	30	317	36%	347	294	895	1	
27	25	Hook Dillon	190	81	University of North Carolina	26	10	18%	22	5	36	0	
28	26	Bob Doll	196	88	University of Colorado	30	120	35%	114	108	315	0	
29	27	Harry Donovan	188	81	Muhlenberg College	23	90	33%	106	38	253	0	
30	28	Dike Eddleman	190	85	University of Illinois at Urbana-Champaign	27	332	37%	260	142	826	1	
31	29	Gene Englund	196	92	University of Wisconsin	32	104	38%	192	41	360	0	
32	30	Bob Evans	188	79	Butler University	24	56	28%	44	55	142	0	
33	31	Johnny Ezersky	190	79	University of Rhode Island	27	143	29%	183	86	413	0	
34	32	Bob Feerick	190	86	Santa Clara University	30	172	34%	174	127	483	0	
35	33	Arnie Ferrin	188	81	University of Utah	24	132	33%	109	95	340	0	
36	34	Jerry Fleishman	188	86	New York University	27	102	29%	151	118	297	0	
37	35	Joe Fuiks*	196	86	Murray State University	28	336	28%	421	56	965	1	
38	36	Elmer Gainer	198	88	DePaul University	31	9	26%	8	7	24	0	
39	37	Harry Gallatin*	198	95	Truman State University	22	263	40%	366	56	803	1	
40	38	Vern Gardner	196	90	University of Utah	24	313	34%	296	119	853	1	
41	39	Bud Grant	190	88	University of Minnesota	22	42	37%	17	19	91	0	
42	40	Don Grate	188	83	Ohio State University	26	1	17%	2	3	4	0	
43	41	Alex Groza	201	98	University of Kentucky	23	521	48%	623	162	1496	1	
44	42	Chick Halbert	206	102	West Texas A&M University	30	108	38%	175	89	328	0	
45	43	Bruce Hale	185	77	Santa Clara University	31	217	35%	285	226	657	1	
46	44	Alex Harnum*	201	95	University of Southern California	26	177	36%	186	129	482	0	
47	45	John Hargis	188	81	University of Texas at Austin	29	223	41%	277	102	643	1	
48	46	Bob Harris	201	88	Oklahoma State University	22	168	36%	223	129	476	0	
49	47	Bob Harrison	201	88	Oklahoma State University	22	125	36%	74	131	300	0	
50	48	Marshall Hawkins	190	92	University of Tennessee	25	55	28%	61	51	152	0	
51	49	Bill Henry	206	97	Rice University	25	89	32%	176	48	296	0	
52	50	Kleggie Hermesen	206	102	University of Minnesota	26	196	32%	247	98	545	1	

.xls

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	id	player	height	weight	college	age	two_points	two_point_p	fta	ast	pts	mayor500	
2	0	Cliff Barker	188	83	University of Kentucky	29	102	37%	106	109	279	0	
3	1	Ralph Beard	178	79	University of Kentucky	22	340	36%	282	233	895	1	
4	2	Charlie Black	196	90	University of Kansas	28	226	28%	321	163	661	1	
5	3	Nelson Bobb	183	77	Temple University	25	80	32%	131	46	242	0	
6	4	Jake Bornheimer	196	90	Muhlenberg College	22	88	29%	117	40	254	0	
7	5	Vince Boryla	196	95	University of Denver	22	204	34%	267	95	612	1	
8	6	Don Boven	193	95	Western Michigan University	24	208	37%	349	137	656	1	
9	7	Harry Boykoff	208	102	St. John's University	27	288	41%	262	149	779	1	
10	8	Joe Bradley	190	79	Oklahoma State University	21	36	27%	38	36	87	0	
11	9	Carl Braun	196	81	Colgate University	22	373	36%	374	247	1031	1	
12	10	Frankie Brian	185	81	Louisiana State University	26	368	32%	488	189	1138	1	
13	11	Price Brookfield	193	83	West Texas A&M University	29	11	48%	13	1	34	0	
14	12	Bob Brown	193	92	Miami University	26	276	36%	252	101	724	1	
15	13	Walt Budko	196	99	Columbia University	24	198	30%	263	146	595	1	
16	14	Tommy Byrnes	190	79	Seton Hall University	26	120	30%	124	88	327	0	
17	15	Bill Calhoun	190	81	City College of San Francisco	22	207	38%	203	115	560	1	
18	16	Don Carlson	183	77	University of Minnesota	30	99	34%	95	76	267	0	
19	17	Leroy Chollet	188	86	Canisius College	24	61	34%	56	37	157	0	
20	18	Bill Closs	196	88	Rice University	28	283	32%	259	160	752	1	
21	19	Jack Coleman	201	88	University of Louisville	25	250	38%	121	153	590	1	
22	20	Jack Cotton	201	90	University of Wyoming	25	97	29%	161	65	276	0	
23	21	Dillard Crocker	193	92	Western Michigan University	25	245	29%	317	85	723	1	
24	22	Chink Crossin	185	74	University of Pennsylvania	26	185	32%	101	148	449	0	
25	23	Fran Curran	183	79	University of Notre Dame	27	98	42%	241	71	395	0	
26	24	Bob Davies*	185	79	Seton Hall University	30	317	36%	347	294	895	1	
27	25	Hook Dillon	190	81	University of North Carolina	26	10	18%	22	5	36	0	
28	26	Bob Doll	196	88	University of Colorado	30	120	35%	114	108	315	0	
29	27	Harry Donovan	188	81	Muhlenberg College	23	90	33%	106	38	253	0	
30	28	Dike Eddieman	190	85	University of Illinois at Urbana-Champaign	27	332	37%	260	142	826	1	
31	29	Gene Englund	196	92	University of Wisconsin	32	104	38%	192	41	360	0	
32	30	Bob Evans	188	79	Butler University	24	56	28%	44	55	142	0	
33	31	Johnny Ezersky	190	79	University of Rhode Island	27	143	29%	183	86	413	0	
34	32	Bob Feerick	190	86	Santa Clara University	30	172	34%	174	127	483	0	
35	33	Arnie Ferrin	188	81	University of Utah	24	132	33%	109	95	340	0	
36	34	Jerry Fleishman	188	86	New York University	27	102	29%	151	118	297	0	
37	35	Joe Fuls*	196	86	Murray State University	28	336	28%	421	56	965	1	
38	36	Elmer Gainer	198	88	DePaul University	31	9	26%	8	7	24	0	
39	37	Harry Gallatin*	198	95	Truman State University	22	263	40%	366	56	803	1	
40	38	Vern Gardner	196	90	University of Utah	24	313	34%	296	119	853	1	
41	39	Bud Grant	190	88	University of Minnesota	22	42	37%	17	19	91	0	
42	40	Don Grate	188	83	Ohio State University	26	1	17%	2	3	4	0	
43	41	Alex Groza	201	98	University of Kentucky	23	521	48%	623	162	1496	1	
44	42	Chick Halbert	206	102	West Texas A&M University	30	108	38%	175	89	328	0	
45	43	Bruce Hale	185	77	Santa Clara University	31	217	35%	285	226	657	1	
46	44	Alex Hannum*	201	95	University of Southern California	26	177	36%	186	129	482	0	
47	45	John Hargis	188	81	University of Texas at Austin	29	223	41%	277	102	643	1	
48	46	Bob Harris	201	88	Oklahoma State University	22	168	36%	223	129	476	0	
49	47	Bob Harrison	201	88	Oklahoma State University	22	125	36%	74	131	300	0	
50	48	Marshall Hawkins	190	92	University of Tennessee	25	55	28%	61	51	152	0	
51	49	Bill Henry	206	97	Rice University	25	89	32%	176	48	296	0	
52	50	Kleggie Hermesen	206	102	University of Minnesota	26	196	32%	247	98	545	1	
53	51	Sonny Hertzberg	178	79	City College of New York	27	275	32%	191	200	693	1	

.CSV

```

id INT,
player VARCHAR,
height INT,
weight INT,
college VARCHAR,
born INT,
birth_city VARCHAR,
birth_state VARCHAR
player_id INT,
year DEC,
position VARCHAR,
age DEC,
Tm VARCHAR,
G VARCHAR,
TS_Percentage DEC,
FTr DEC,
OWS DEC,
DWS DEC,
WS DEC,
FG DEC,
FGA DEC,
FG_Percentage DEC,
Two_Points DEC,
Two_PA DEC,
Two_Point_Percentage DEC,
eFG_Percentage DEC,
FT DEC,
FTA DEC,
FT_Percentage DEC,
AST DEC,
PF DEC,
PTS DEC

```

3. al 8. Posteriormente utilizando COLAB se cargó dicha base de datos y tomando como referencia el EJERCICIO 7 (gracias por compartir), se ejecutó de la siguiente forma.

1. Importar base de datos y librerías.

```
1: IMPORTAR LIBRERIAS/DATASETS Y REALIZAR ANÁLISIS EXPLORATORIO DE DATOS
```

```
[1] from google.colab import drive
drive.mount('/content/drive')
Mounted at /content/drive
```

```
[2] from google.colab import drive
drive.mount('/content/drive') #Debemos confirmar acceso a nuestra cuenta para el COLAB
#print(drive.mount) #Imprime una dirección hexadecimal del apuntador a la carpeta drive
%cd "/content/drive/MyDrive/4DatosPractica"
%pwd
%ls
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
'21 SEP FINAL.csv' jugador.csv ROC.png UCI_Credit_Card.csv
'3 OCT EJERCICIO FINAL PARA PYTHON.csv' player.csv telecom_churn.csv
```

```
[3] import numpy as np # Manipulación de arrays Multi-dimensionales
import pandas as pd # Manipulación de DataFrames
import matplotlib.pyplot as plt # Visualización de Datos
import seaborn as sns # Visualización de Datos
from sklearn import metrics
```

```
[4] # Leer el fichero CSV
jugador_df = pd.read_csv("/content/jugador.csv")
```

```
[5] # Cargar las 5 primeras observaciones
jugador_df.head()
```

	id	player	height	weight	college	age	two_points	two_point_percentage	fta	ast	pts	mayor500
0	0	Cliff Barker	188	83	University of Kentucky	29	102	37%	106	109	279	0
1	1	Ralph Beard	178	79	University of Kentucky	22	340	36%	282	233	895	1
2	2	Charlie Black	196	90	University of Kansas	28	226	28%	321	163	661	1
3	3	Nelson Bobb	183	77	Temple University	25	80	32%	131	46	242	0
4	4	Jake Bornheimer	196	90	Muhlenberg College	22	88	29%	117	40	254	0

2. Revisar base de datos, 12 columnas y 3070 filas.
3. Revisar el tipo de columnas, 3 object y 9 int64.
4. Obtener datos estadísticos de la base de datos, como son, valores min y max, mediana, desviación estándar, cuantiles.
5. Eliminar columnas object.

```
[7] # Comprobar la dimensión del dataframe
jugador_df.shape
(3070, 12)
```

```
[8] # Mostrar las columnas de las características
jugador_df.columns
Index(['id', 'player', 'height', 'weight', 'college', 'age', 'two_points', 'two_point_percentage', 'fta', 'ast', 'pts', 'mayor500'], dtype='object')
```

```
[9] # Obtener el resumen de los tipos de datos del dataframe
jugador_df.dtypes
```

	o
id	int64
player	object
height	int64
weight	int64
college	object
age	int64
two_points	int64
two_point_percentage	object
fta	int64
ast	int64
pts	int64
mayor500	int64

dtype: object

+ Código + Texto

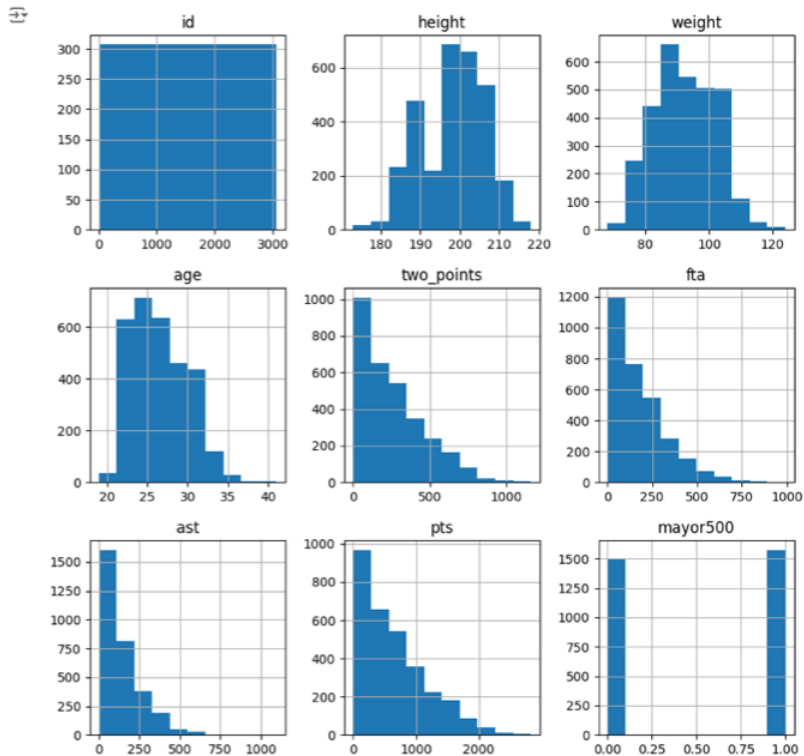
```
[10] # Desplegar los detalles estadísticos del dataframe  
jugador_df.describe()
```

	id	height	weight	age	two_points	fta	ast	pts	mayor500
count	3070.000000	3070.000000	3070.000000	3070.000000	3070.000000	3070.000000	3070.000000	3070.000000	3070.000000
mean	1534.500000	198.041694	92.515309	26.445603	253.669055	174.998371	136.940717	639.156678	0.512704
std	886.376989	8.198711	9.203127	3.313451	209.493498	149.675414	126.829546	526.162488	0.499920
min	0.000000	173.000000	68.000000	19.000000	0.000000	1.000000	0.000000	0.000000	0.000000
25%	767.250000	193.000000	86.000000	24.000000	81.000000	56.000000	39.000000	203.250000	0.000000
50%	1534.500000	198.000000	93.000000	26.000000	207.500000	138.000000	102.500000	515.500000	1.000000
75%	2301.750000	203.000000	99.000000	29.000000	375.000000	254.750000	200.000000	946.750000	1.000000
max	3069.000000	218.000000	124.000000	41.000000	1159.000000	991.000000	1099.000000	2831.000000	1.000000

6. Visualización con histogramas de las 9 columnas.
7. Mostrar mediante un gráfico circular jugadores con y sin 500 pts anotados.
8. Obtener matriz de correlaciones de jugadores.

2: VISUALIZACION DE DATOS

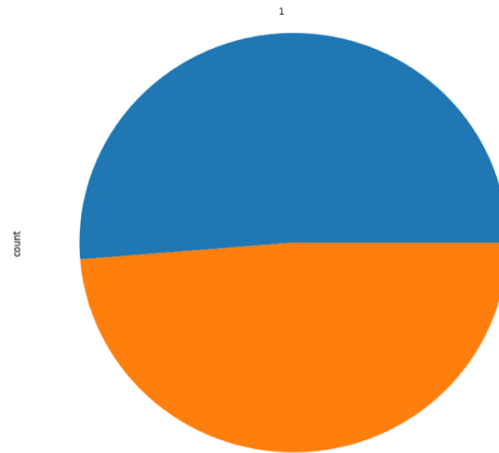
```
[11] jugador_df.hist(figsize = (10, 10))  
plt.show()
```



```
[12] jugador_df["mayor500"].value_counts()

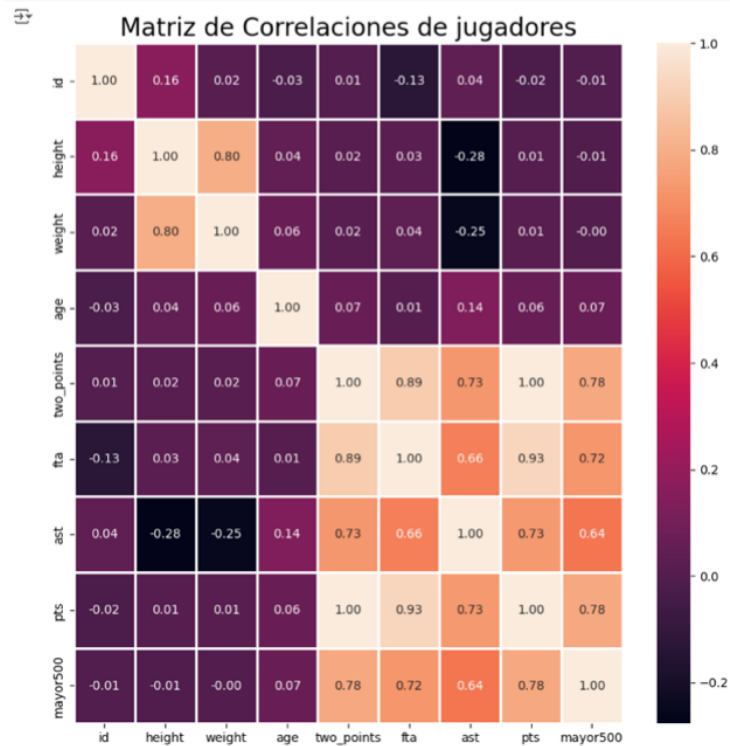
count
mayor500
1      1574
0      1496
dtype: int64

[13] plt.figure(figsize = [10, 10])
jugador_df["mayor500"].value_counts().plot(kind='pie')
<Axes: ylabel='count'>
```



```
[15] # Matriz de Jugadores
corr_matrix = jugador_df.corr()
plt.figure(figsize = [10, 10])
cm = sns.heatmap(corr_matrix,linewidths = 1,annot = True,fmt = ".2f")
plt.title("Matriz de Correlaciones de jugadores", fontsize = 20)
plt.show()

# Se muestra que "pts" tiene altas correlacionados.
```



9. Seleccionar y eliminar columna mayor500, asimismo, eliminar columna id.
10. División y entrenamiento, Xtrain, Xtest, ytrain, ytest. Test size del 40%.

```
✓ [173] # Las características innecesarias disminuirían la velocidad de entrenamiento, la interpretabilidad del modelo y el rendimiento de la generalización en los datos de prueba.
0.1 # Por lo tanto, es crucial encontrar y seleccionar las características más útiles en el conjunto de datos.
# Asignación de características de entrada a X y de salida a y
X = jugador_df.drop(["mayor500", "id"], axis = "columns") # area_code y phone_number son características irrelevantes para continuar con el entrenamiento del modelo
y = jugador_df["mayor500"]

✓ [174] X.shape
0.1 #Tenemos menos elementos..
(3070, 7)

✓ [175] X
0.1
height weight age two_points fta ast pts
0 188 83 29 102 106 109 279
1 178 79 22 340 282 233 895
2 196 90 28 226 321 163 661
3 183 77 25 80 131 46 242
4 196 90 22 88 117 40 254
... ..
3065 198 90 25 209 119 184 514
3066 198 90 26 220 114 228 538
3067 198 90 27 253 159 251 641
3068 198 90 28 170 104 125 427
3069 198 90 29 34 19 34 83
3070 rows x 7 columns

Proximos pasos: Generar código con X Ver gráficos recomendados New interactive sheet

✓ [176] y.shape
0.1 (3070,)

✓ [177] # Realizar la división de entrenamiento/prueba
0.1 from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.4, random_state = 1228) #40% para pruebas y el resto 60% entrenamiento

✓ [178] X_train.shape
0.1 (1842, 7)
```

11. Con randomForest, se calculan y grafican 7 variables.

```
✓ [180] from sklearn.ensemble import RandomForestClassifier #Algoritmos de conjunto/ensamblado
0.1
rf = RandomForestClassifier()
rf.fit(X_train, y_train.values.ravel()) #que lo entrene y ajuste para predecir Y

✓ [182] # Representar la importancia de las características, ¿cuales son las variables importantes?
0.1 feat_scores= pd.DataFrame({"Fracción de las variables afectadas" : rf.feature_importances_, index = X.columns}) #Que tanto aporta esa variable al modelo
feat_scores= feat_scores.sort_values(by = "Fracción de las variables afectadas") #Ordenalas según la importancia.
feat_scores.plot(kind = "barh", figsize = (10, 5)) #hacemos un grafico de barras horizontal de la mas a la menos importante
sns.despine() #limita el eje x y al contenido

pts
two_points
fta
ast
age
height
weight
Fracción de las variables afectadas
```

El gráfico anterior es generado por el algoritmo Random Forest El gráfico indica que "pts" encabeza la lista de características importantes seguido de "two_points" y así sucesivamente.

12. Utilizando clasificador regresión logística. Importamos librería, modelos, y predict y son 1228 pruebas arroja una exactitud de 0.99. Asimismo, se obtiene y muestra matriz de confusión.

```
183] from sklearn.linear_model import LogisticRegression
      from sklearn.metrics import classification_report, confusion_matrix

      modelo_LR = LogisticRegression()

      modelo_LR.fit(X_train, y_train)

⚡ /usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:469: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
  https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
  https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
n_iter_i = _check_optimize_result(
  LogisticRegression()
LogisticRegression()

✓ [184] y_predict = modelo_LR.predict(X_test)

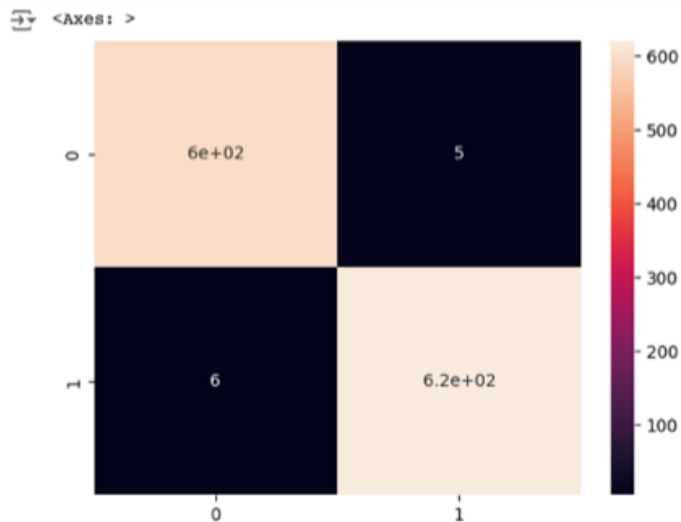
[187] print(classification_report(y_test, y_predict))
# La precisión es la relación TP/(TP+FP)
# TP = Verdaderos Positivos
# FP = Falsos Positivos
# FN = Falsos Negativos
# El recall (recuperación) es la relación entre TP/(TP+FN)
# La puntuación F-beta puede interpretarse como una media armónica ponderada de la precisión y la recuperación.
# donde una puntuación F-beta alcanza su mejor valor en 1 y su peor puntuación en 0.

⚡
```

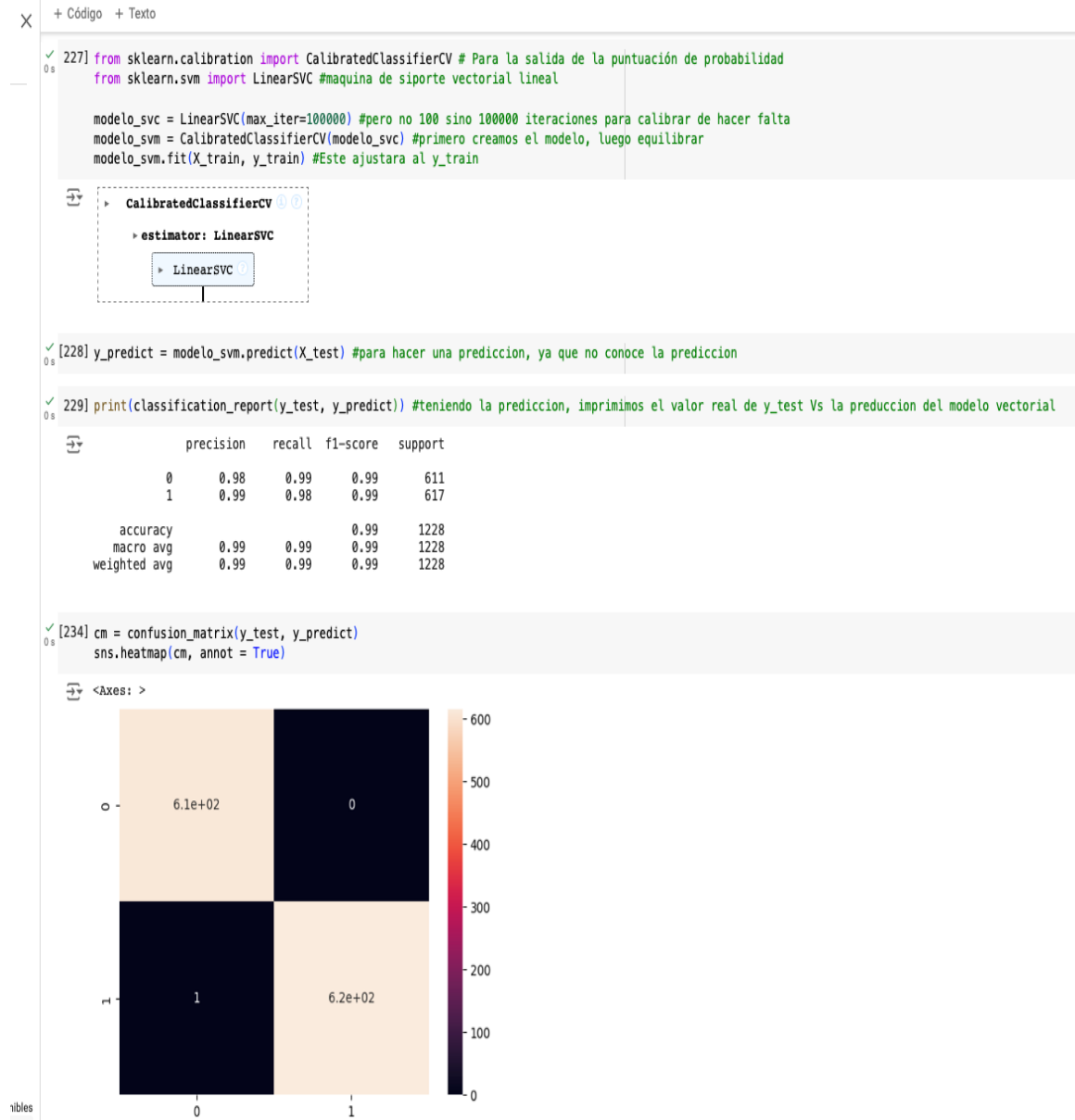
	precision	recall	f1-score	support
0	0.99	0.99	0.99	601
1	0.99	0.99	0.99	627
accuracy			0.99	1228
macro avg	0.99	0.99	0.99	1228
weighted avg	0.99	0.99	0.99	1228

- Imprime la matriz de confusión y comenta los resultados

```
[188] cm = confusion_matrix(y_test, y_predict)
      sns.heatmap(cm, annot = True)
```



13. Utilizando maquina soporte vectorial. Importamos librería, modelos, ypredict y son 1228 pruebas arroja una exactitud de 0.99. Asimismo, se obtiene y muestra matriz de confusión.



14. Utilizando clasificador bosque aleatorio. Importamos librería, modelos, y predict y son 1228 pruebas arroja una exactitud de 1.0. Asimismo, se obtiene y muestra matriz de confusión.

```
[192] from sklearn.ensemble import RandomForestClassifier
      modelo_rf = RandomForestClassifier() #por default trabaja con 100 arboles/estimadores
      modelo_rf.fit(X_train, y_train) #apliquemos el método de entrenamiento
```

RandomForestClassifier

RandomForestClassifier()

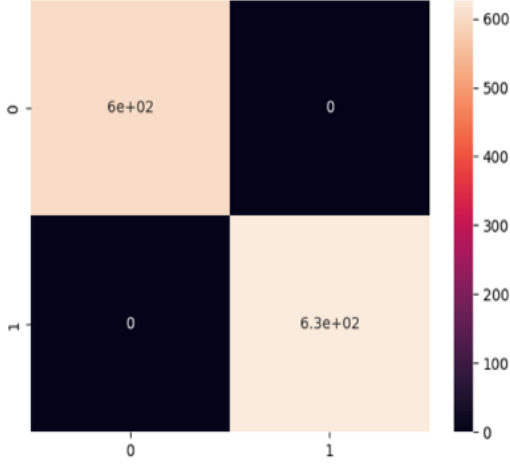
```
[193] y_predict = modelo_rf.predict(X_test)
```

```
[194] print(classification_report(y_test, y_predict))
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	601
1	1.00	1.00	1.00	627
accuracy			1.00	1228
macro avg	1.00	1.00	1.00	1228
weighted avg	1.00	1.00	1.00	1228

```
[195] cm = confusion_matrix(y_test, y_predict)
      sns.heatmap(cm, annot = True)
```

<Axes: >



	Actual 0	Actual 1
Predicted 0	601	0
Predicted 1	0	627

15. Utilizando K-nearest Neighbour (KNN). Importamos librería, modelos, ypredict y son 1228 pruebas arroja una exactitud de 0.98. Asimismo, se obtiene y muestra matriz de confusión.

```
196] from sklearn.neighbors import KNeighborsClassifier
```

```
modelo_knn = KNeighborsClassifier() #por default trabaja con 5 vecinos.  
modelo_knn.fit(X_train, y_train)
```

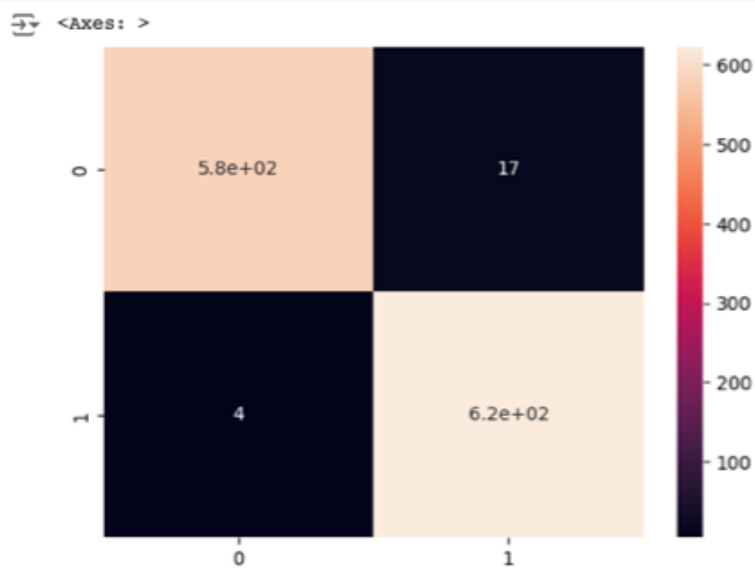
```
▼ KNeighborsClassifier  
KNeighborsClassifier()
```

```
✓ [197] y_predict = modelo_knn.predict(X_test)  
0 s
```

```
✓ [198] print(classification_report(y_test, y_predict))  
0 s
```

```
precision    recall  f1-score   support  
  
0           0.99      0.97      0.98        601  
1           0.97      0.99      0.98        627  
  
accuracy          0.98  
macro avg          0.98      0.98      0.98        1228  
weighted avg       0.98      0.98      0.98        1228
```

```
✓ [199] cm = confusion_matrix(y_test, y_predict)  
0 s      sns.heatmap(cm, annot = True)
```



16. Utilizando Naive Bayes Gaussian. Importamos librería, modelos, ypredict y son 1228 pruebas arroja una exactitud de 0.97. Asimismo, se obtiene y muestra matriz de confusión.

```
✓ [200] from sklearn.naive_bayes import GaussianNB
```

```
✓ [201] modelo_gnb = GaussianNB()  
0 s modelo_gnb.fit(X_train, y_train)
```

↗ GaussianNB 3 0
GaussianNB()

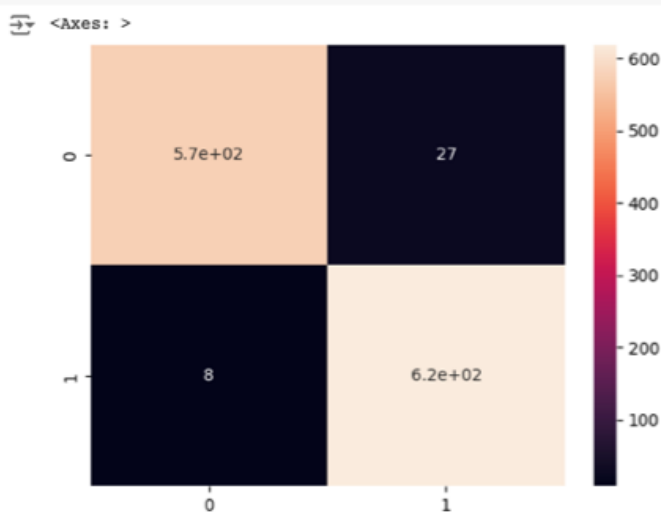
```
✓ [202] y_predict = modelo_gnb.predict(X_test)
```

```
✓ [203] print(classification_report(y_test, y_predict))
```

↗

	precision	recall	f1-score	support
0	0.99	0.96	0.97	601
1	0.96	0.99	0.97	627
accuracy			0.97	1228
macro avg	0.97	0.97	0.97	1228
weighted avg	0.97	0.97	0.97	1228

```
✓ [204] cm = confusion_matrix(y_test, y_predict)  
0 s sns.heatmap(cm, annot = True)
```



18. Comparación de modelos:

19. Se muestran las curvas ROC.

20. Conclusiones.

Comprueba este enlace: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html

```
array([[1.00000000e+00, 9.54732760e-27],
       [1.00000000e+00, 2.23330089e-26],
       [1.00000000e+00, 3.02329823e-44],
       ...,
       [1.00000000e+00, 4.84972555e-17],
       [0.00000000e+00, 1.00000000e+00],
       [0.00000000e+00, 1.00000000e+00]])
```

```
[207] y_test
```

```

# Import pandas as pd
import pandas as pd

# Create the DataFrame 'mayor500'
mayor500 = pd.DataFrame({'name': names, 'votes': votes})

# Print the DataFrame
print(mayor500)

```

✓ [209] fpr1

```
>> array([0., 0., 0., 0., 0.,  
         0., 0., 0., 0., 0.,  
         0.00166389, 0.00166389, 0.00332779, 0.00332779, 0.00831947,  
         0.00831947, 0.01164725, 0.01164725, 0.01830283, 0.01830283,  
         0.01996672, 0.01996672, 0.0266223 , 0.0266223 , 0.03993344,  
         0.03993344, 1.])
```

```
array([0.008633796, 0.80781754, 0.81188223, 0.81339713,
        0.81818182, 0.82137161, 0.82296655, 0.826156, 0.98485104,
        0.98405104, 0.98724083, 0.98724083, 0.99043062, 0.99843062,
        0.99202552, 0.99202552, 0.99362041, 0.99362041, 0.99521531,
        0.99521531, 0.99681021, 0.99681021, 0.9984051, 0.9984051,
        1., 1.])
```

```
array([ inf, 1.00000000e+00, 1.00000000e+00, 1.00000000e+00,
        1.00000000e+00, 1.00000000e+00, 1.00000000e+00, 1.00000000e+00,
        1.00000000e+00, 9.97932850e-01, 9.97111006e-01, 9.85216298e-01,
        9.74558007e-01, 9.73793965e-01, 5.51737405e-01, 4.81587644e-01,
        3.56725587e-01, 3.35514479e-01, 2.23956293e-01, 2.16859870e-01,
        2.14756977e-01, 1.91943667e-01, 8.98086061e-02, 7.98356972e-02,
        1.89064198e-02, 1.05198088e-02, 4.90036609e-03])
```

```
fp1, tp1, thresh1 = roc_curve(y_test, modelo_LR.predict_proba(X_test)[: , 1], pos_label = 1)
fp2, tp2, thresh2 = roc_curve(y_test, modelo svm.predict_proba(X_test)[: , 1], pos_label = 1)
fp3, tp3, thresh3 = roc_curve(y_test, modelo_rf.predict_proba(X_test)[: , 1], pos_label = 1)
fp4, tp4, thresh4 = roc_curve(y_test, modelo_knn.predict_proba(X_test)[: , 1], pos_label = 1)
fp5, tp5, thresh5 = roc_curve(y_test, modelo_gnb.predict_proba(X_test)[: , 1], pos_label = 1)
```

```

✓ [213] # Puntuación AUC
0.8
from sklearn.metrics import roc_auc_score

auc_score1 = roc_auc_score(y_test, modelo_lr.predict_proba(X_test)[:, 1])
auc_score2 = roc_auc_score(y_test, modelo_svm.predict_proba(X_test)[:, 1])
auc_score3 = roc_auc_score(y_test, modelo_rf.predict_proba(X_test)[:, 1])
auc_score4 = roc_auc_score(y_test, modelo_knn.predict_proba(X_test)[:, 1])
auc_score5 = roc_auc_score(y_test, modelo_gnb.predict_proba(X_test)[:, 1])

print("Regresión Logística: ", auc_score1) # Regresión Logística
print("Máquina de Soporte Vectorial: ", auc_score2) # Máquina de Soporte Vectorial
print("Bosque Aleatorio: ", auc_score3) # Bosque Aleatorio
print("K-Nearest Neighbors: ", auc_score4) # K-Nearest Neighbors
print("Naive Bayes: ", auc_score5) # Naive Bayes

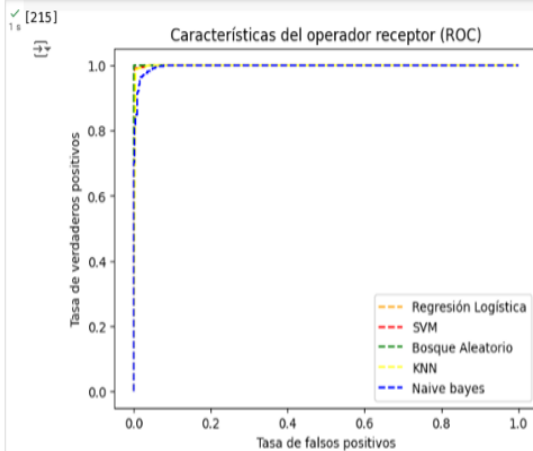
→ Regresión Logística: 0.999785047249799
Máquina de Soporte Vectorial: 0.999745241184947
Bosque Aleatorio: 1.0
K-Nearest Neighbors: 0.9972268441486412
Naive Bayes: 0.9966483293394581

✓ [215]
1.8
plt.plot(fpr1, tpr1, linestyle = "--", color = "orange", label = "Regresión Logística")
plt.plot(fpr2, tpr2, linestyle = "--", color = "red", label = "SVM")
plt.plot(fpr3, tpr3, linestyle = "--", color = "green", label = "Bosque Aleatorio")
plt.plot(fpr4, tpr4, linestyle = "--", color = "yellow", label = "KNN")
plt.plot(fpr5, tpr5, linestyle = "--", color = "blue", label = "Naive bayes")

plt.title('Características del operador receptor (ROC)')
plt.xlabel('Tasa de falsos positivos')
plt.ylabel('Tasa de verdaderos positivos')

plt.legend(loc = 'best')
plt.savefig('ROC', dpi = 300)
plt.show()

```



✓ CONCLUSIONES

El gráfico muestra que el algoritmo Random Forest obtuvo el mejor AUC. Por lo tanto, está claro que este modelo realizó un mejor trabajo a la hora de clasificar a jugadores con más de 500 pts.

```

✓ [215]
0.8
y_predict = modelo_rf.predict(X_test)
print(classification_report(y_test, y_predict))

→

```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	601
1	1.00	1.00	1.00	627
accuracy			1.00	1228
macro avg	1.00	1.00	1.00	1228
weighted avg	1.00	1.00	1.00	1228

CONCLUSIONES Y RECOMENDACIONES

Como resultado de la ejecución de 5 modelos para el análisis de la base de datos de jugadores de una liga universitaria de basketball.

- a) Regresión Logística: 0.999785047249799
- b) Máquina de Soporte Vectorial: 0.999745241184947
- c) Bosque Aleatorio: 1.0
- d) K-Nearest Neighbors: 0.9972268441486412
- e) Naive Bayes: 0.9966483293394581

Se concluye que el modelo de bosque aleatorio es el que mejor describe este análisis. Los jugadores que logran más de 500 pts.