Predecir cuantos juegos puede ganar en una temporada un equipo específico de béisbol de las Ligas Mayores de Béisbol

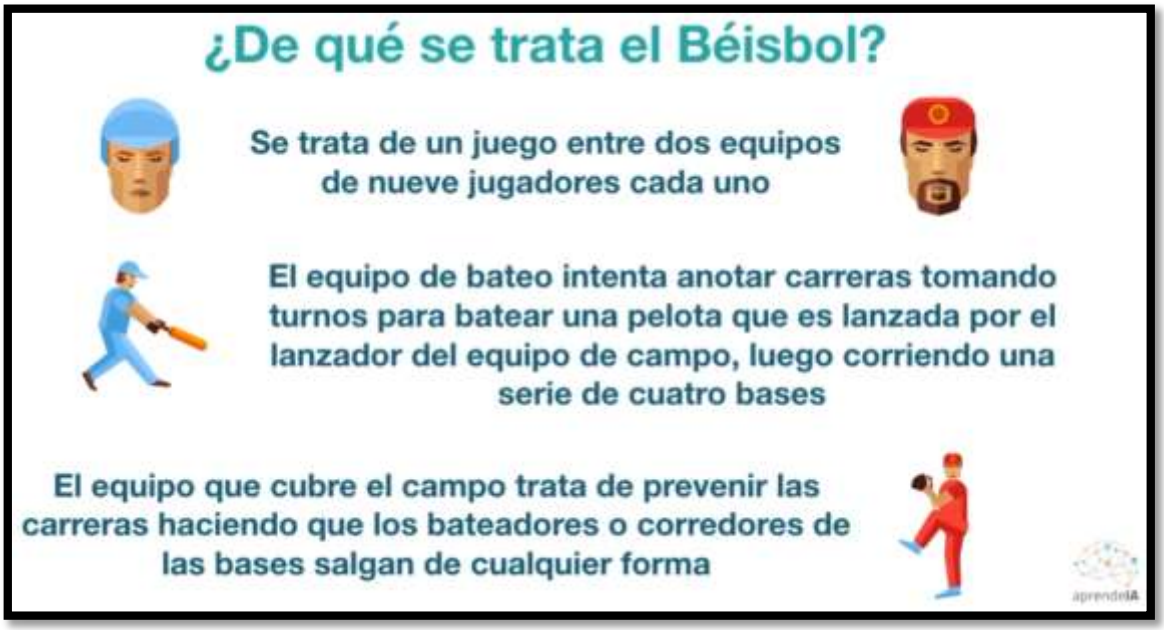
Enlace github: https://github.com/dvgr78/PROYECTOS-MODELOS-MACHINE-LEARNING/tree/main/MODELO%20PARTIDOS%20BASEBALL

email: dvgr78@hotmail.com

Contenido

TÍTULO

Predecir cuantos juegos puede ganar en una temporada un equipo específico de béisbol de las Ligas Mayores de Béisbol



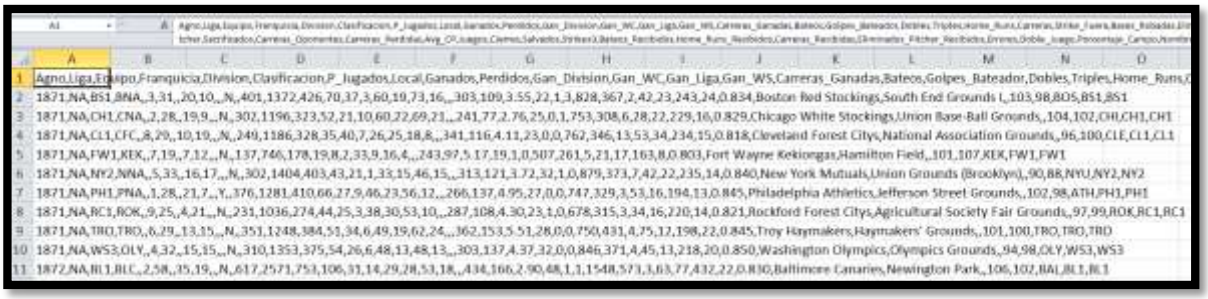
LIBRERÍAS

```
# Importamos librerías
import pandas as pd
import matplotlib.pyplot as plt
```

ANÁLISIS DE DATOS

dataframe

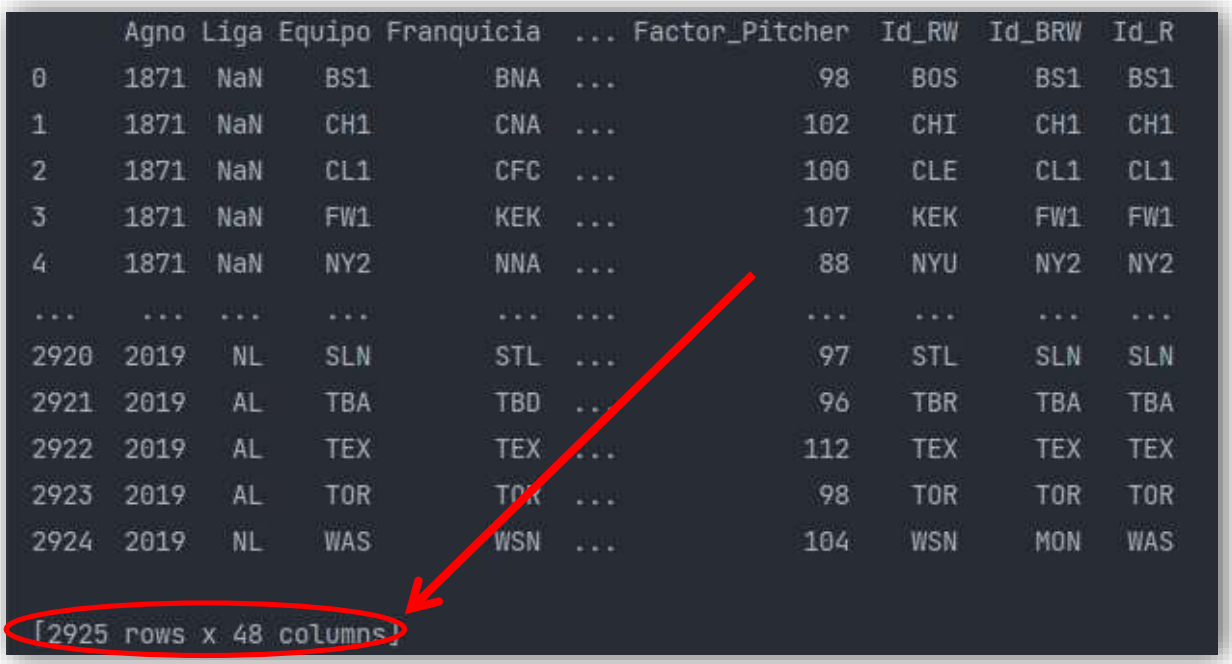
En primer lugar, como científicos de datos, deberemos investigar sobre los datos a los que vamos a aplicar un modelo o algoritmo para utilizar el más conveniente.



Una primera impresión sobre el archivo “Team.csv” refleja la complejidad de los datos disponibles, aun estando de forma estructurada o tabular.

Sigamos investigando:

```
# Importamos los datos contenidos en Team.csv
data = pd.read_csv('Teams.csv')
print(data)
```



	Agno	Liga	Equipo	Franquicia	...	Factor_Pitcher	Id_RW	Id_BRW	Id_R
0	1871	NaN	BS1	BNA	...	98	BOS	BS1	BS1
1	1871	NaN	CH1	CNA	...	102	CHI	CH1	CH1
2	1871	NaN	CL1	CFC	...	100	CLE	CL1	CL1
3	1871	NaN	FW1	KEK	...	107	KEK	FW1	FW1
4	1871	NaN	NY2	NNA	...	88	NYU	NY2	NY2
...	...	...	...	...	...	...	...	...	...
2920	2019	NL	SLN	STL	...	97	STL	SLN	SLN
2921	2019	AL	TBA	TBD	...	96	TBR	TBA	TBA
2922	2019	AL	TEX	TEX	...	112	TEX	TEX	TEX
2923	2019	AL	TOR	TOR	...	98	TOR	TOR	TOR
2924	2019	NL	WAS	WSN	...	104	WSN	MON	WAS

[2925 rows x 48 columns]

head y shape

```
# Analizamos los datos
print(data.shape)
```

```
(2925, 48)
```

Mediante una vista preliminar y mediante el uso del método ***.shape*** podemos empezar a extraer alguna información. Disponemos de 2925 registros informados y un total de 48 características, por lo que a priori deberemos tener en cuenta si son necesarias para nuestro modelo todas las características pues podríamos caer y un sobreajuste o incluir mucho "ruido" que distorsione nuestras predicciones y modelo.

```
print(data.head(15))
```

	Agno	Liga	Equipo	Franquicia	...	Factor_Pitcher	Id_RW	Id_BRW	Id_R
0	1871	NaN	BS1	BNA	...	98	BOS	BS1	BS1
1	1871	NaN	CH1	CNA	...	102	CHI	CH1	CH1
2	1871	NaN	CL1	CFC	...	100	CLE	CL1	CL1
3	1871	NaN	FW1	KEK	...	107	KEK	FW1	FW1
4	1871	NaN	NY2	NNA	...	88	NYU	NY2	NY2
5	1871	NaN	PH1	PNA	...	98	ATH	PH1	PH1
6	1871	NaN	RC1	ROK	...	99	ROK	RC1	RC1
7	1871	NaN	TRO	TRO	...	100	TRO	TRO	TRO
8	1871	NaN	WS3	OLY	...	98	OLY	WS3	WS3
9	1872	NaN	BL1	BLC	...	102	BAL	BL1	BL1
10	1872	NaN	BR1	ECK	...	96	ECK	BR1	BR1
11	1872	NaN	BR2	BRA	...	122	BRA	BR2	BR2
12	1872	NaN	BS1	BNA	...	100	BOS	BS1	BS1
13	1872	NaN	CL1	CFC	...	100	CLE	CL1	CL1
14	1872	NaN	MID	MAN	...	103	MAN	MID	MID

[15 rows x 48 columns]

Mediante el método ***.head(15)*** hemos visualizado los 15 primeros registros de nuestros datos y podemos extraer algunas conclusiones.

La característica "Liga" está vacía "NaN" por lo que su utilidad aparente mente es nula.

Existen muchas características "no numéricas" o categóricas por lo que podríamos normalizarlas o nos pueden servir como campos de "traducción" para poder extraer conclusiones, sin embargo, si tuviéramos que realizar una predicción numérica carecerían de interés alguno.

dtypes

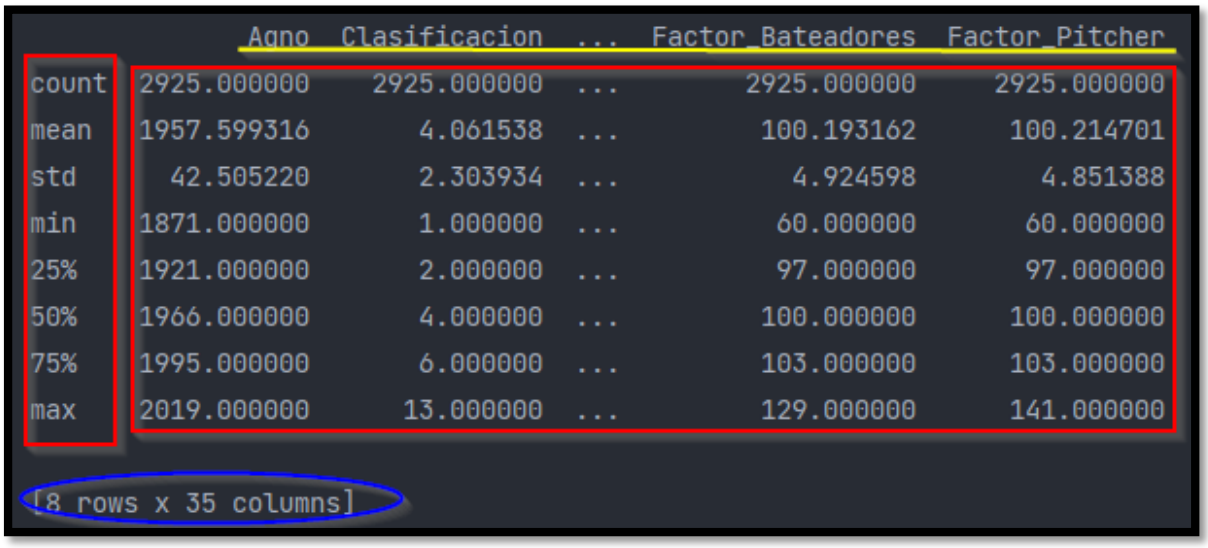
```
# Formato de los datos
print(data.dtypes)
```

```
Agno                    int64
Liga                   object
Equipo                 object
Franquicia             object
Division               object
Clasificacion           int64
P_Jugados               int64
Local                 float64
Ganados                 int64
Perdidos                int64
Gan_Division           object
Gan_WC                 object
Gan_Liga               object
Gan_WS                 object
Carreras_Ganadas        int64
Bateos                  int64
Golpes_Bateador         int64
Dobles                  int64
Triples                 int64
Home_Runs               int64
Carreras              float64
Strike_Fuera          float64
Bases_Robadas         float64
Eliminados            float64
Eliminados_Pitcher    float64
Sacrificados          float64
Carreras_Oponentes      int64
Carreras_Perdidas       int64
Avg_CP                float64
Juegos                  int64
Cierres                 int64
Salvados                int64
Strikex3                int64
Bateos_Recibidos        int64
Home_Runs_Recibidos     int64
Carreras_Recibidas      int64
Eliminados_Pitcher_Recibidos    int64
Errores                 int64
Doble_Juego             int64
Porcentaje_Campo      float64
Nombre_Equipo          object
Nombre_Estadio         object
Asistencia_Estadio    float64
Factor_Bateadores       int64
Factor_Pitcher          int64
Id_RW                  object
Id_BRW                 object
Id_R                   object
dtype: object
```

Mediante ***.dtypes*** visualizaremos los tipos de datos de cada una de las características, llamando profundamente la atención el tipo “object” y que a grandes rasgos nos indica que se trata de características no numéricas. Deberemos tenerlo en cuenta en fases posteriores.

describe

```
# Descripción de los datos
print(data.describe())
```



	Agno	Clasificacion	...	Factor_Bateadores	Factor_Pitcher
count	2925.000000	2925.000000	...	2925.000000	2925.000000
mean	1957.599316	4.061538	...	100.193162	100.214701
std	42.505220	2.303934	...	4.924598	4.851388
min	1871.000000	1.000000	...	60.000000	60.000000
25%	1921.000000	2.000000	...	97.000000	97.000000
50%	1966.000000	4.000000	...	100.000000	100.000000
75%	1995.000000	6.000000	...	103.000000	103.000000
max	2019.000000	13.000000	...	129.000000	141.000000

[8 rows x 35 columns]

Mediante ***.describe()*** podemos hacer visibles magnitudes y métricas estadísticas de cada una de las características como contar registros, media, varianza, máximo, mínimo y percentiles. Obviamente para aquellas características categóricas no dispondremos de esta información y vemos su disminución:

"[… rows x 35 columns]" vs. "[… rows x 48 columns]".

¿Casualidad 13 columnas tipo "object"? Empieza a tomar forma nuestro modelo.

Aspectos interesantes:

- Primer año 1871
- Último año 2019

isnull

```
# Buscamos datos nulos o faltantes
print(data.isnull().sum())
```

```
Agno                    0
Liga                   50
Equipo                  0
Franquicia              0
Division             1517
Clasificacion           0
P_Jugados               0
Local                 399
Ganados                 0
Perdidos                0
Gan_Division         1545
Gan_WC               2181
Gan_Liga               28
Gan_WS                357
Carreras_Ganadas        0
Bateos                  0
Golpes_Bateador         0
Dobles                  0
Triples                 0
Home_Runs               0
Carreras                1
```

```
Strike_Fuera            16
Bases_Robadas          126
Eliminados             832
Eliminados_Pitcher    1158
Sacrificados          1541
Carreras_Oponentes       0
Carreras_Perdidas        0
Avg_CP                   0
Juegos                   0
Cierres                  0
Salvados                 0
Strikex3                 0
Bateos_Recibidos         0
Home_Runs_Recibidos      0
Carreras_Recibidas       0
Eliminados_Pitcher_Recibidos  0
Errores                  0
Doble_Juego              0
Porcentaje_Campo         0
Nombre_Equipo            0
Nombre_Estadio          34
```

```
Asistencia_Estadio    279
Factor_Bateadores       0
Factor_Pitcher          0
Id_RW                   0
Id_BRW                  0
Id_R                    0
dtype: int64
```

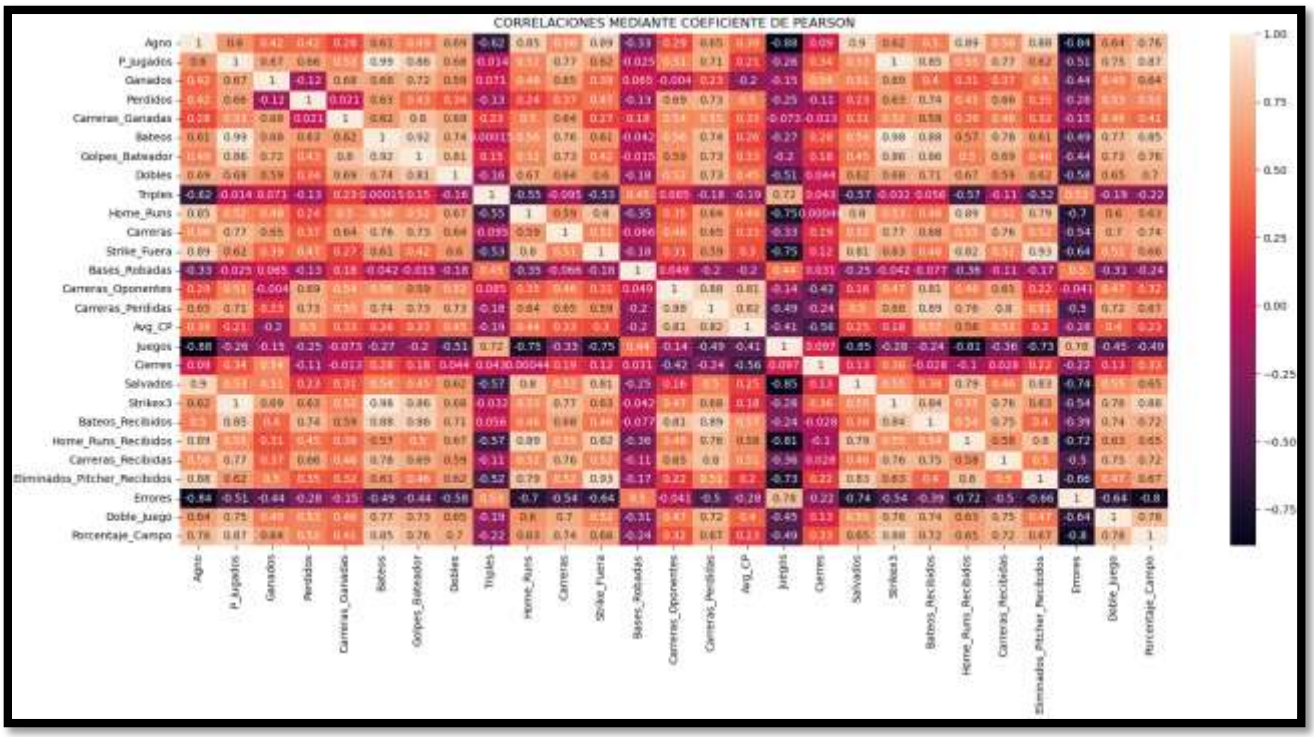
Mediante ***isnull().sum()*** contaremos aquellos registros nulos de cada una de la características o atributos, lo cual nos puede ser útil para procesamiento posteriores e incluso para descartar alguna característica. He marcado algunas características donde la cantidad de registros informados nulos es destacable por su posible impacto en nuestro futuro modelo. En la etapa de procesamiento de datos deberemos decidir qué hacer con ellos en base al impacto y teniendo en cuenta diferentes aspectos.

CORRELACIONES

Pearson

```
#   Coeficiente de pearson +- >0.75 descartado
pearson      = data.corr(method="pearson")
print(pearson)
plt.figure(figsize=(14, 8))
pearson.style.background_gradient(cmap='coolwarm')
sns.heatmap(pearson, annot=True)
plt.show()
```

	Agno	...	Porcentaje_Campo
Agno	1.000000	...	0.759478
P_Jugados	0.602292	...	0.870324
Ganados	0.415507	...	0.644892
Perdidos	0.421635	...	0.523580
Carreras_Ganadas	0.284747	...	0.413431
Bateos	0.605383	...	0.853567
Golpes_Bateador	0.494803	...	0.761612
Dobles	0.690776	...	0.702055
Triples	-0.622958	...	-0.215250
Home_Runs	0.846526	...	0.627844

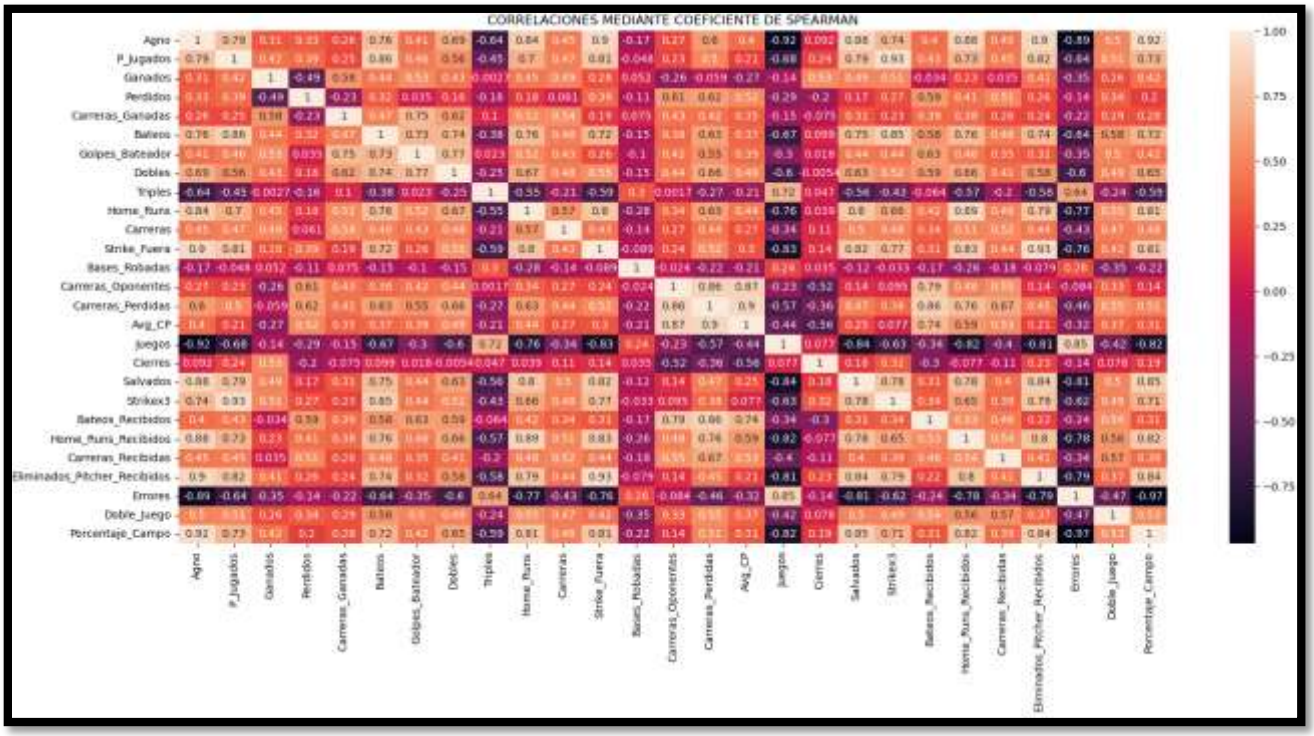


Mediante el ***coeficiente de Pearson*** podemos extraer la relación lineal existente entre cada una de las características y según el valor se encuentre > + 0.75 ó < - 0.75 nos obligará a realizar algunas transformaciones.

Spearman

```
#   Coeficiente de spearman +- >0.75 descartado
spearman    = data.corr(method= "spearman", min_periods = 4)
print(spearman)
plt.figure(figsize=(14, 8))
spearman.style.background_gradient(cmap='coolwarm')
sns.heatmap(spearman, annot=True)
plt.show()
```

	Agno	...	Porcentaje_Campo
Agno	1.000000	...	0.918954
P_Jugados	0.785383	...	0.732611
Ganados	0.310607	...	0.421388
Perdidos	0.334837	...	0.202033
Carreras_Ganadas	0.256051	...	0.276798
Bateos	0.763914	...	0.719731
Golpes_Bateador	0.405188	...	0.416025
Dobles	0.686147	...	0.646674
Triples	-0.641775	...	-0.590865
Home_Runs	0.835446	...	0.810136



Mediante el ***coeficiente de Spearman*** podemos extraer la relación lineal existente entre cada una de las características y según el valor se encuentre > + 0.75 ó < -0.75 nos obligará a realizar algunas transformaciones.

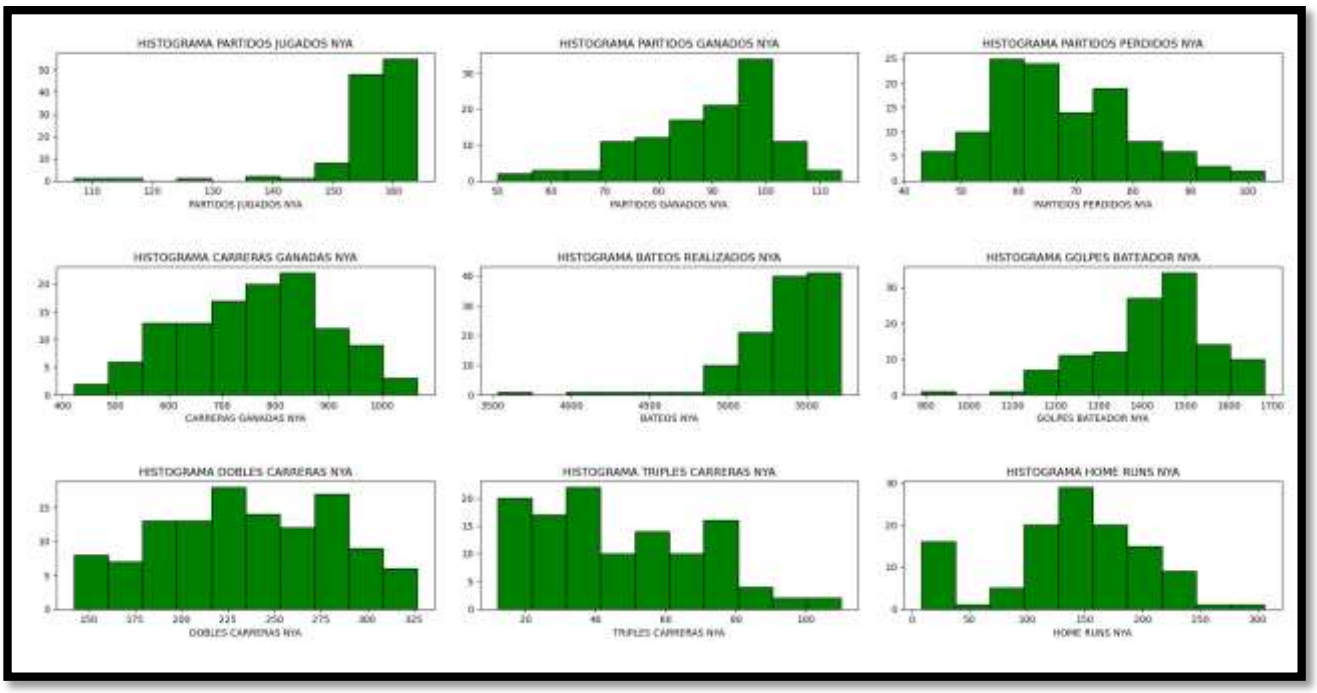
Asimetría o Sesgo

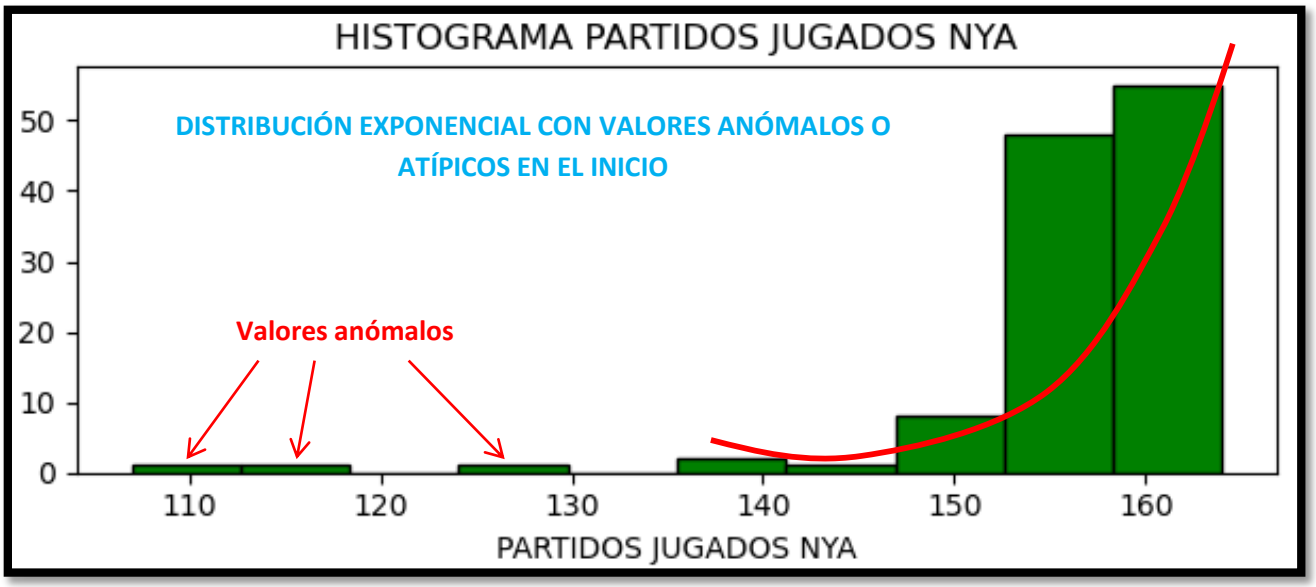
Agno	-0.352491
P_Jugados	-3.384415
Ganados	-1.097926
Perdidos	-0.797680
Carreras_Ganadas	-0.705870
Bateos	-3.384733
Golpes_Bateador	-2.434393
Dobles	-0.634881
Triples	0.915272
Home_Runs	0.090229

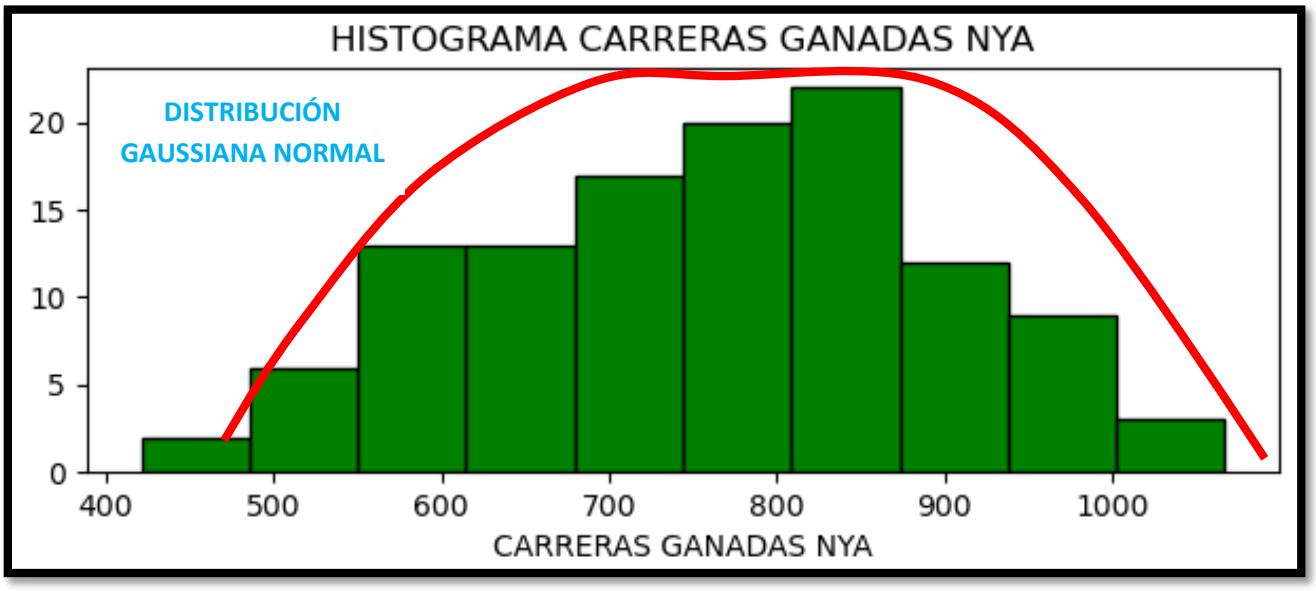
La asimetría o seso será otra métrica que deberemos tener en cuenta para fases posteriores como el procesamiento de los datos.

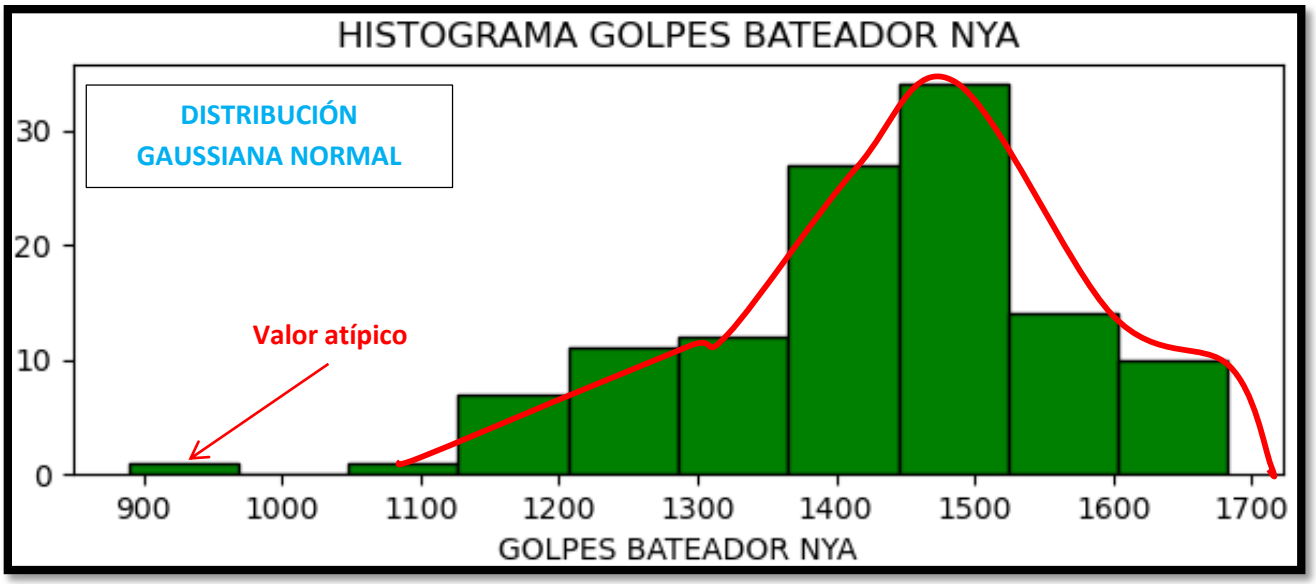
ANÁLISIS VISUAL DE LA DISTRIBUCIÓN DE LOS DATOS

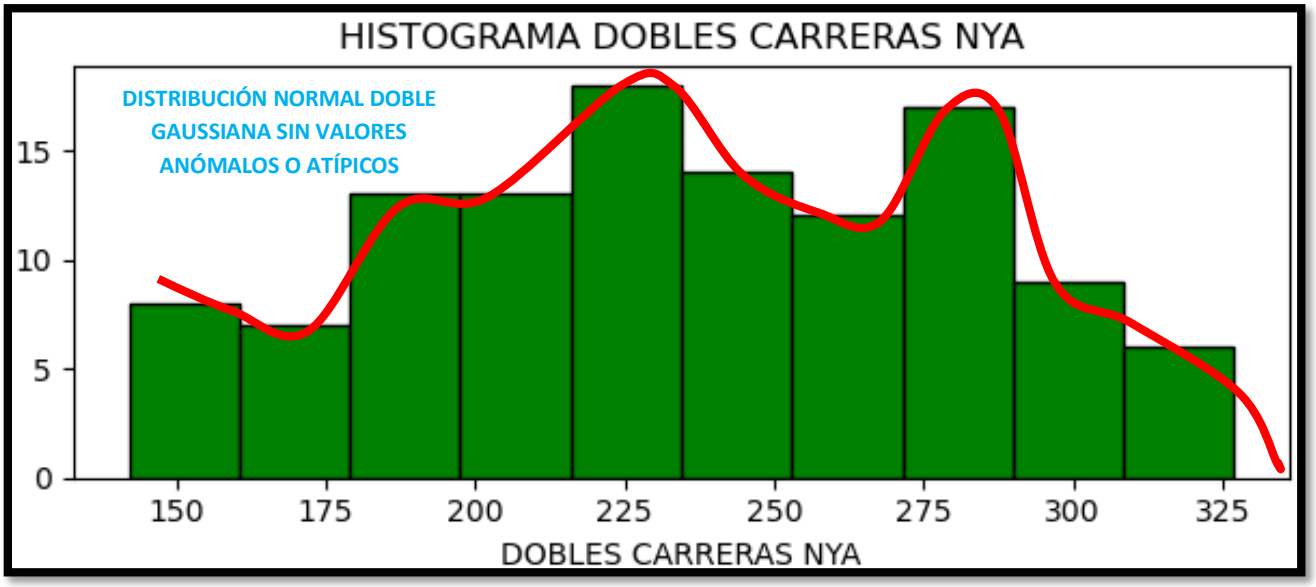
A primera vista y como ejemplo, visualizaremos la distribución de las 9 primeras características, y haremos una primera valoración.






HISTOGRAMA CARRERAS GANADAS NYA
DISTRIBUCIÓN
GAUSSIANA NORMAL
20
15
10
5
0
400
500
600
700
800
900
1000
CARRERAS GANADAS NYA


HISTOGRAMA GOLPES BATEADOR NYA
DISTRIBUCIÓN
GAUSSIANA NORMAL
Valor atípico
30
20
10
0
900
1000
1100
1200
1300
1400
1500
1600
1700
GOLPES BATEADOR NYA


HISTOGRAMA DOBLES CARRERAS NYA
DISTRIBUCIÓN NORMAL DOBLE
GAUSSIANA SIN VALORES
ANÓMALOS O ATÍPICOS
15
10
5
0
150
175
200
225
250
275
300
325
DOBLES CARRERAS NYA

PROCESAMIENTO DE LOS DATOS

```
# Eliminación de columnas innecesarias
borrar_columnas = ['Liga','Franquicia','Clasificacion','Local',
                   'Gan_Division','Gan_WC','Gan_Liga','Gan_WS',
                   'Sacrificados','Nombre_Equipo','Nombre_Estadio',
                   'Asistencia_Estadio','Factor_Bateadores',
                   'Factor_Pitcher','Id_R','Id_BRW','Id_RW']
data = data.drop(borrar_columnas, axis=1)
data = data.drop(['Eliminados','Eliminados_Pitcher','Division'],
axis=1)
# Completamos los datos nulos con la media de cada uno
data['Carreras']        =
data['Carreras'].fillna(data['Carreras'].median())
data['Strike_Fuera']    =
data['Strike_Fuera'].fillna(data['Strike_Fuera'].median())
data['Bases_Robadas']   =
data['Bases_Robadas'].fillna(data['Bases_Robadas'].median())
print(data.shape)
print(data.dtypes)
print(data.isnull().sum())
```

```
(2925, 28)
```

```
Agno                    int64
Equipo                 object
P_Jugados               int64
Ganados                 int64
Perdidos                int64
Carreras_Ganadas        int64
Bateos                  int64
Golpes_Bateador         int64
Dobles                  int64
Triples                 int64
Home_Runs               int64
Carreras              float64
Strike_Fuera          float64
Bases_Robadas         float64
```

```
Carreras_Oponentes              int64
Carreras_Perdidas               int64
Avg_CP                        float64
Juegos                          int64
Cierres                         int64
Salvados                        int64
Strikex3                        int64
Bateos_Recibidos                int64
Home_Runs_Recibidos             int64
Carreras_Recibidas              int64
Eliminados_Pitcher_Recibidos    int64
Errores                         int64
Doble_Juego                     int64
Porcentaje_Campo              float64
```

Agno	0
Equipo	0
P_Jugados	0
Ganados	0
Perdidos	0
Carreras_Ganadas	0
Bateos	0
Golpes_Bateador	0
Dobles	0
Triples	0
Home_Runs	0
Carreras	0
Strike_Fuera	0
Bases_Robadas	0

Carreras_Oponentes	0
Carreras_Perdidas	0
Avg_CP	0
Juegos	0
Cierres	0
Salvados	0
Strikex3	0
Bateos_Recibidos	0
Home_Runs_Recibidos	0
Carreras_Recibidas	0
Eliminados_Pitcher_Recibidos	0
Errores	0
Doble_Juego	0
Porcentaje_Campo	0

Segmentación de los datos

Por eras o etapas:

Se dividirá en 8 eras en total:

Menor a 1920 = Era 1
Entre 1920 y 1941 = Era 2
Entre 1942 y 1945 = Era 3
Entre 1946 y 1962 = Era 4
Entre 1963 y 1976 = Era 5
Entre 1977 y 1992 = Era 6
Entre 1993 y 2009 = Era 7
Mayor a 2010 = Era 8

```
# Separamos por eras o etapas
i =1
for year in data['Agno']:
    if year < 1920:
        data.loc[i,"era"]=1
    elif year >= 1920 and year <= 1941:
        data.loc[i, "era"] = 2
    elif year >= 1942 and year <= 1945:
        data.loc[i, "era"] = 3
    elif year >= 1946 and year <= 1962:
        data.loc[i, "era"] = 4
    elif year >= 1963 and year <= 1976:
```

```
        data.loc[i, "era"] = 5
    elif year >= 1977 and year <= 1992:
        data.loc[i, "era"] = 6
    elif year >= 1993 and year <= 2009:
        data.loc[i, "era"] = 7
    elif year >= 2010:
        data.loc[i, "era"] = 8
    i += 1
```

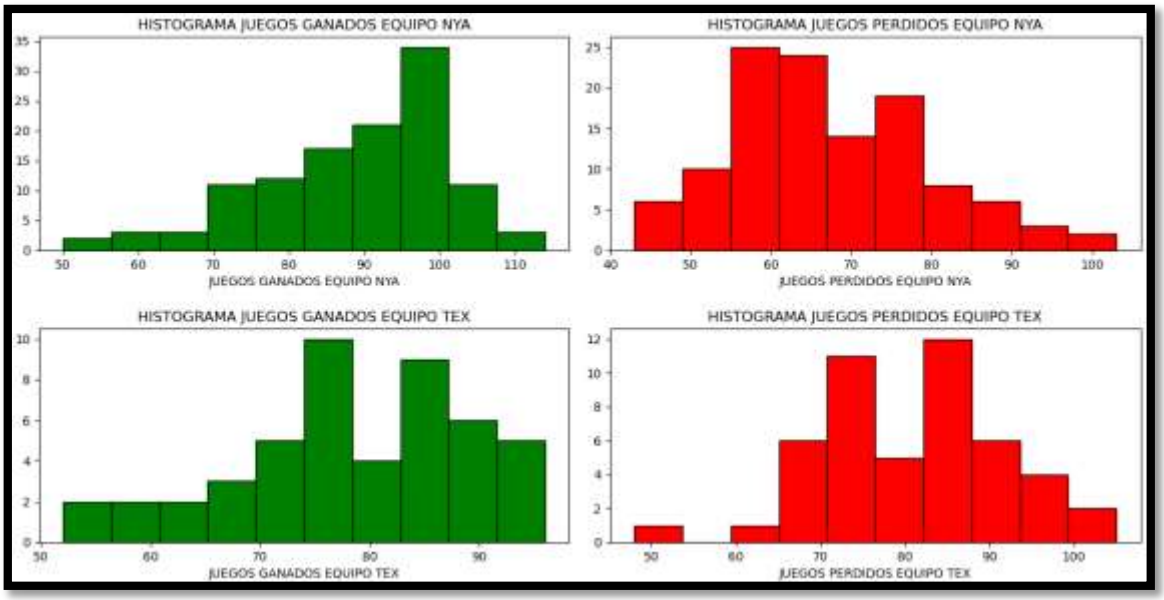
Por décadas:


```
# Separamos por décadas
j =1
for year in data['Agno']:
    if year     <   1920:
        data.loc[j,"decada"]    =   1910

    elif year   >=  1920 and year   <=  1929:
        data.loc[j, "decada"]   =   1920

    elif year   >=  1930 and year   <=  1939:
        data.loc[j, "decada"]   =   1930

    elif year   >=  1940 and year   <=  1949:
        data.loc[j, "decada"]   =   1940

    elif year   >=  1950 and year   <=  1959:
        data.loc[j, "decada"]   =   1950

    elif year   >=  1960 and year   <=  1969:
        data.loc[j, "decada"]   =   1960

    elif year   >=  1970 and year   <=  1979:
        data.loc[j, "decada"]   =   1970

    elif year   >=  1980 and year   <=  1989:
        data.loc[j, "decada"]   =   1980

    elif year   >=  1990 and year   <=  1999:
        data.loc[j, "decada"]   =   1990

    elif year   >=  2000 and year   <=  2009:
        data.loc[j, "decada"]   =   2000
```

```
    elif year   >=  2010:
        data.loc[j, "decada"]   =   2010

    j += 1
```

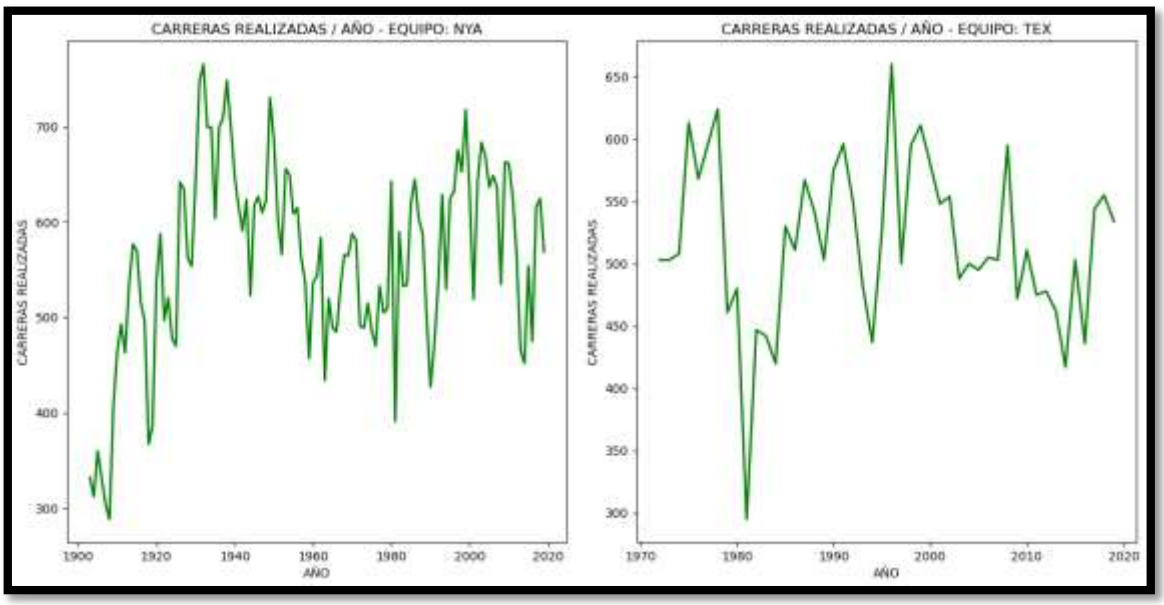
Visualización gráfica

```
# Graficamos datos
fig = plt.figure(figsize=(12,6))
ax1 = fig.add_subplot(2,2,1)
ax2 = fig.add_subplot(2,2,2)
ax3 = fig.add_subplot(2,2,3)
ax4 = fig.add_subplot(2,2,4)

ax1.hist(data_equipo['Ganados'],bins=10, color='green',linewidth=1,
edgecolor="black")
ax1.set_xlabel("JUEGOS GANADOS EQUIPO {}".format(equipo))
ax1.set_title("HISTOGRAMA JUEGOS GANADOS EQUIPO {}".format(equipo))

ax2.hist(data_equipo['Perdidos'],bins=10, color='red', linewidth=1,
edgecolor="black")
ax2.set_xlabel("JUEGOS PERDIDOS EQUIPO {}".format(equipo))
ax2.set_title("HISTOGRAMA JUEGOS PERDIDOS EQUIPO {}".format(equipo))

ax3.hist(data_equipo1['Ganados'],bins=10, color='green', linewidth=1,
edgecolor="black")
ax3.set_xlabel("JUEGOS GANADOS EQUIPO {}".format(equipo1))
ax3.set_title("HISTOGRAMA JUEGOS GANADOS EQUIPO {}".format(equipo1))

ax4.hist(data_equipo1['Perdidos'],bins=10, color='red',linewidth=1,
edgecolor="black")
ax4.set_xlabel("JUEGOS PERDIDOS EQUIPO {}".format(equipo1))
ax4.set_title("HISTOGRAMA JUEGOS PERDIDOS EQUIPO {}".format(equipo1))
plt.tight_layout()
plt.show()
```

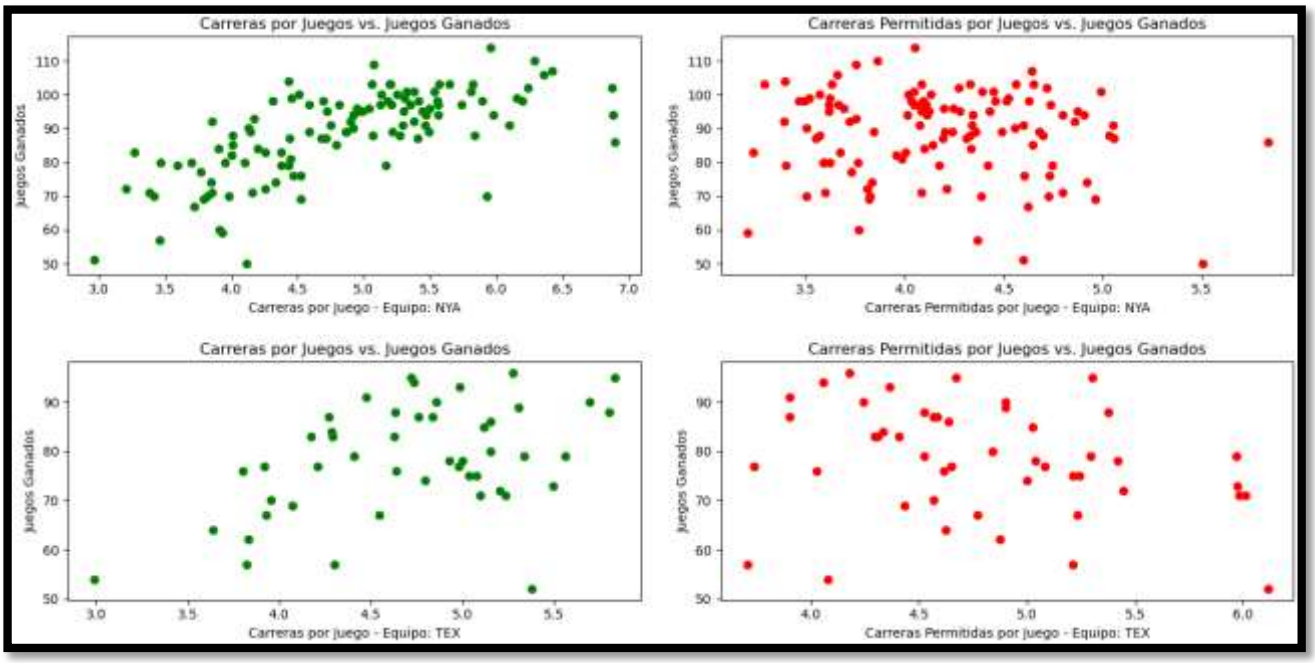



```
# Graficamos las carreras realizadas
fig = plt.figure(figsize=(12,6))
ax1 = fig.add_subplot(1,2,1)
ax2 = fig.add_subplot(1,2,2)
ax1.plot(data_equipo['Agno'],data_equipo['Carreras'],linewidth=2.0,color="green")
```

```
ax1.set_title("CARRERAS REALIZADAS / AÑO - EQUIPO: {}".format(equipo))
ax1.set_ylabel("CARRERAS REALIZADAS")
ax1.set_xlabel("AÑO")
ax2.plot(data_equipo1['Agno'],data_equipo1['Carreras'],linewidth=2.0,color="green")
ax2.set_title("CARRERAS REALIZADAS / AÑO - EQUIPO: {}".format(equipo1))
ax2.set_xlabel("AÑO")
ax2.set_ylabel("CARRERAS REALIZADAS")
plt.tight_layout()
plt.show()
```




```
# Gráficas adicionales
CarrerasG_x_Partido  = data_equipo['Carreras_Ganadas'] / data_equipo['P_Jugados']
CarrerasG_x_Partido1 = data_equipo1['Carreras_Ganadas'] / data_equipo1['P_Jugados']
CarrerasP_x_Partido = data_equipo['Carreras_Oponentes'] / data_equipo['P_Jugados']
CarrerasP_x_Partido1 = data_equipo1['Carreras_Oponentes'] /
data_equipo1['P_Jugados']

fig = plt.figure(figsize=(12,6))
ax1 = fig.add_subplot(2,2,1)
ax2 = fig.add_subplot(2,2,2)
ax3 = fig.add_subplot(2,2,3)
ax4 = fig.add_subplot(2,2,4)
ax1.scatter(CarrerasG_x_Partido,data_equipo['Ganados'], c="green")
ax1.set_title("Carreras por Juegos vs. Juegos Ganados")
ax1.set_ylabel("Juegos Ganados")
ax1.set_xlabel("Carreras por Juego - Equipo: {}".format(equipo))
ax2.scatter(CarrerasP_x_Partido,data_equipo['Ganados'], c="red")
ax2.set_title("Carreras Permitidas por Juegos vs. Juegos Ganados")
ax2.set_ylabel("Juegos Ganados")
ax2.set_xlabel("Carreras Permitidas por Juego - Equipo: {}".format(equipo))
ax3.scatter(CarrerasG_x_Partido1,data_equipo1['Ganados'], c="green")
ax3.set_title("Carreras por Juegos vs. Juegos Ganados")
ax3.set_ylabel("Juegos Ganados")
ax3.set_xlabel("Carreras por Juego - Equipo: {}".format(equipo1))
ax4.scatter(CarrerasP_x_Partido1,data_equipo1['Ganados'], c="red")
ax4.set_title("Carreras Permitidas por Juegos vs. Juegos Ganados")
ax4.set_ylabel("Juegos Ganados")
ax4.set_xlabel("Carreras Permitidas por Juego - Equipo: {}".format(equipo1))
plt.tight_layout()
plt.show()
```



Entrenamiento de los modelos

```
# Eliminamos últimas columnas que no son necesarias
data_equipo = data_equipo.drop(['Agno','Equipo'], axis = 1)
data_equipo1 = data_equipo1.drop(['Agno','Equipo'], axis = 1)

# Importamos nuevas librerías para entrenar nuestro modelo
from sklearn.model_selection    import train_test_split
from sklearn.linear_model       import LinearRegression
from sklearn.metrics            import r2_score
from sklearn.metrics            import mean_squared_error

# Definimos las variables independientes y la dependiente
X = data_equipo.drop("Ganados",axis = 1)
y = data_equipo["Ganados"]
X1 = data_equipo1.drop("Ganados",axis = 1)
y1 = data_equipo1["Ganados"]

# Separamos entre entrenamiento y prueba
X_train, X_test, y_train, y_test        = train_test_split(X, y, test_size=0.20,
random_state=1)
X_train1, X_test1, y_train1, y_test1    = train_test_split(X1, y1, test_size=0.20,
random_state=1)

# Elegimos el modelo "Regresión Lineal"
algoritmo = LinearRegression()
algoritmo1 = LinearRegression()

# Entrenamos el algoritmo
algoritmo.fit(X_train, y_train)
algoritmo1.fit(X_train1, y_train1)

# Realizamos una predicción
y_test_pred = algoritmo.predict(X_test)
y_test_pred1 = algoritmo1.predict(X_test1)

# Calculamos la precisión del modelo
# Error promedio al cuadrado
# Calculo de R2
```

```
mse =mean_squared_error(y_test,y_test_pred)
rmse_rf =(mean_squared_error(y_test,y_test_pred))**(1/2)
r2 = r2_score(y_test,y_test_pred)
mse1 =mean_squared_error(y_test1,y_test_pred1)
rmse_rf1 =(mean_squared_error(y_test1,y_test_pred1))**(1/2)
r21 = r2_score(y_test1,y_test_pred1)
print("************************************************************")
print("**********       MSE {}:                      {}".format(equipo, mse))
print("**********       ERROR CUADRÁTICO MEDIO {}: {}".format(equipo, rmse_rf))
print("**********       R2 {}:                       {}".format(equipo, r2))
print("**********       MSE {}:                      {}".format(equipo1, mse1))
print("**********       ERROR CUADRÁTICO MEDIO {}: {}".format(equipo1, rmse_rf1))
print("**********       R2 {}:                       {}".format(equipo1, r21))
print("************************************************************")
```

```
************************************************************
**********       MSE NYA:                    0.4376403884470463
**********       ERROR CUADRÁTICO MEDIO NYA: 0.6615439429448706
**********       R2 NYA:                     0.9955653918840072
**********       MSE TEX:                    0.28840036545715286
**********       ERROR CUADRÁTICO MEDIO TEX: 0.5370292035421843
**********       R2 TEX:                     0.995453966496577
************************************************************
```

Código completo:

```
# Importamos librerías
import pandas as pd
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# Importamos los datos contenidos en Team.csv
data = pd.read_csv('Teams.csv')
print(data)

# ************************************
# *****   ANÁLISIS DE LOS DATOS    *****
# ************************************
# Analizamos los datos
print(data.shape)
# Formato de los datos
print(data.dtypes)
# Buscamos datos nulos o faltantes
print(data.isnull().sum())

# ************************************
# ***** PROCESAMIENTO DE LOS DATOS *****
# ************************************

# Eliminación de columnas innecesarias
borrar_columnas = ['Liga','Franquicia','Clasificacion','Local',
                   'Gan_Division','Gan_WC','Gan_Liga','Gan_WS',
                   'Sacrificados','Nombre_Equipo','Nombre_Estadio',
```

```
                  'Asistencia_Estadio','Factor_Bateadores',
                  'Factor_Pitcher','Id_R','Id_BRW','Id_RW']
data = data.drop(borrar_columnas, axis=1)
data = data.drop(['Eliminados','Eliminados_Pitcher','Division'], axis=1)

# Completamos los datos nulos con la media de cada uno
data['Carreras']        = data['Carreras'].fillna(data['Carreras'].median())
data['Strike_Fuera']    = data['Strike_Fuera'].fillna(data['Strike_Fuera'].median())
data['Bases_Robadas']   = data['Bases_Robadas'].fillna(data['Bases_Robadas'].median())
print(data.shape)
print(data.dtypes)
print(data.isnull().sum())

# Separamos por eras o etapas
i =0
for year in data['Agno']:
    if year < 1920:
        data.loc[i,"era"]=1
    elif year >= 1920 and year <= 1941:
        data.loc[i, "era"] = 2
    elif year >= 1942 and year <= 1945:
        data.loc[i, "era"] = 3
    elif year >= 1946 and year <= 1962:
        data.loc[i, "era"] = 4
    elif year >= 1963 and year <= 1976:
        data.loc[i, "era"] = 5
    elif year >= 1977 and year <= 1992:
        data.loc[i, "era"] = 6
    elif year >= 1993 and year <= 2009:
        data.loc[i, "era"] = 7
    elif year >= 2010:
        data.loc[i, "era"] = 8
    i += 1

# Separamos por décadas
j =0
for year in data['Agno']:
    if year     <   1920:
        data.loc[j,"decada"]    =   1910

    elif year   >=  1920 and year   <=  1929:
        data.loc[j, "decada"]   =   1920

    elif year   >=  1930 and year   <=  1939:
        data.loc[j, "decada"]   =   1930

    elif year   >=  1940 and year   <=  1949:
        data.loc[j, "decada"]   =   1940

    elif year   >=  1950 and year   <=  1959:
        data.loc[j, "decada"]   =   1950

    elif year   >=  1960 and year   <=  1969:
        data.loc[j, "decada"]   =   1960

    elif year   >=  1970 and year   <=  1979:
        data.loc[j, "decada"]   =   1970

    elif year   >=  1980 and year   <=  1989:
        data.loc[j, "decada"]   =   1980

    elif year   >=  1990 and year   <=  1999:
```

```
        data.loc[j, "decada"]   =   1990

    elif year   >=  2000 and year   <=  2009:
        data.loc[j, "decada"]   =   2000

    elif year   >=  2010:
        data.loc[j, "decada"]   =   2010

    j += 1
print(data.Equipo)

# Seleccionamos un equipo para evaluar
equipo = 'NYA'
data_equipo = data.loc[data.Equipo == equipo]
equipo1 = 'TEX'
data_equipo1 = data.loc[data.Equipo == equipo1]

# ************************************
# *****   VISUALIZACIÓN GRÁFICA    *****
# ************************************
# Graficamos datos
fig = plt.figure(figsize=(12,6))
ax1 = fig.add_subplot(2,2,1)
ax2 = fig.add_subplot(2,2,2)
ax3 = fig.add_subplot(2,2,3)
ax4 = fig.add_subplot(2,2,4)

ax1.hist(data_equipo['Ganados'],bins=10, color='green',linewidth=1, edgecolor="black")
ax1.set_xlabel("JUEGOS GANADOS EQUIPO {}".format(equipo))
ax1.set_title("HISTOGRAMA JUEGOS GANADOS EQUIPO {}".format(equipo))

ax2.hist(data_equipo['Perdidos'],bins=10, color='red', linewidth=1, edgecolor="black")
ax2.set_xlabel("JUEGOS PERDIDOS EQUIPO {}".format(equipo))
ax2.set_title("HISTOGRAMA JUEGOS PERDIDOS EQUIPO {}".format(equipo))

ax3.hist(data_equipo1['Ganados'],bins=10, color='green', linewidth=1, edgecolor="black")
ax3.set_xlabel("JUEGOS GANADOS EQUIPO {}".format(equipo1))
ax3.set_title("HISTOGRAMA JUEGOS GANADOS EQUIPO {}".format(equipo1))

ax4.hist(data_equipo1['Perdidos'],bins=10, color='red',linewidth=1, edgecolor="black")
ax4.set_xlabel("JUEGOS PERDIDOS EQUIPO {}".format(equipo1))
ax4.set_title("HISTOGRAMA JUEGOS PERDIDOS EQUIPO {}".format(equipo1))
plt.tight_layout()
plt.show()

# Graficamos las carreras realizadas
fig = plt.figure(figsize=(12,6))
ax1 = fig.add_subplot(1,2,1)
ax2 = fig.add_subplot(1,2,2)
ax1.plot(data_equipo['Agno'],data_equipo['Carreras'],linewidth=2.0,color="green")
ax1.set_title("CARRERAS REALIZADAS / AÑO - EQUIPO: {}".format(equipo))
ax1.set_ylabel("CARRERAS REALIZADAS")
ax1.set_xlabel("AÑO")
ax2.plot(data_equipo1['Agno'],data_equipo1['Carreras'],linewidth=2.0,color="green")
ax2.set_title("CARRERAS REALIZADAS / AÑO - EQUIPO: {}".format(equipo1))
ax2.set_xlabel("AÑO")
ax2.set_ylabel("CARRERAS REALIZADAS")
plt.tight_layout()
plt.show()

# Gráficas adicionales
CarrerasG_x_Partido  = data_equipo['Carreras_Ganadas'] / data_equipo['P_Jugados']
```

```
CarrerasG_x_Partido1 = data_equipo1['Carreras_Ganadas'] / data_equipo1['P_Jugados']
CarrerasP_x_Partido = data_equipo['Carreras_Oponentes'] / data_equipo['P_Jugados']
CarrerasP_x_Partido1 = data_equipo1['Carreras_Oponentes'] / data_equipo1['P_Jugados']

fig = plt.figure(figsize=(12,6))
ax1 = fig.add_subplot(2,2,1)
ax2 = fig.add_subplot(2,2,2)
ax3 = fig.add_subplot(2,2,3)
ax4 = fig.add_subplot(2,2,4)
ax1.scatter(CarrerasG_x_Partido,data_equipo['Ganados'], c="green")
ax1.set_title("Carreras por Juegos vs. Juegos Ganados")
ax1.set_ylabel("Juegos Ganados")
ax1.set_xlabel("Carreras por Juego - Equipo: {}".format(equipo))
ax2.scatter(CarrerasP_x_Partido,data_equipo['Ganados'], c="red")
ax2.set_title("Carreras Permitidas por Juegos vs. Juegos Ganados")
ax2.set_ylabel("Juegos Ganados")
ax2.set_xlabel("Carreras Permitidas por Juego - Equipo: {}".format(equipo))
ax3.scatter(CarrerasG_x_Partido1,data_equipo1['Ganados'], c="green")
ax3.set_title("Carreras por Juegos vs. Juegos Ganados")
ax3.set_ylabel("Juegos Ganados")
ax3.set_xlabel("Carreras por Juego - Equipo: {}".format(equipo1))
ax4.scatter(CarrerasP_x_Partido1,data_equipo1['Ganados'], c="red")
ax4.set_title("Carreras Permitidas por Juegos vs. Juegos Ganados")
ax4.set_ylabel("Juegos Ganados")
ax4.set_xlabel("Carreras Permitidas por Juego - Equipo: {}".format(equipo1))
plt.tight_layout()
plt.show()

# Eliminamos últimas columnas que no son necesarias
data_equipo = data_equipo.drop(['Agno','Equipo'], axis = 1)
data_equipo1 = data_equipo1.drop(['Agno','Equipo'], axis = 1)

# Importamos nuevas librerías para entrenar nuestro modelo
from sklearn.model_selection    import train_test_split
from sklearn.linear_model       import LinearRegression
from sklearn.metrics            import r2_score
from sklearn.metrics            import mean_squared_error

# Definimos las variables independientes y la dependiente
X = data_equipo.drop("Ganados",axis = 1)
y = data_equipo["Ganados"]
X1 = data_equipo1.drop("Ganados",axis = 1)
y1 = data_equipo1["Ganados"]

# Separamos entre entrenamiento y prueba
X_train, X_test, y_train, y_test        = train_test_split(X, y, test_size=0.20,
random_state=1)
X_train1, X_test1, y_train1, y_test1    = train_test_split(X1, y1, test_size=0.20,
random_state=1)

# Elegimos el modelo "Regresión Lineal"
algoritmo = LinearRegression()
algoritmo1 = LinearRegression()

# Entrenamos el algoritmo
algoritmo.fit(X_train, y_train)
algoritmo1.fit(X_train1, y_train1)

# Realizamos una predicción
y_test_pred = algoritmo.predict(X_test)
y_test_pred1 = algoritmo1.predict(X_test1)
```

```
# Calculamos la precisión del modelo
# Error promedio al cuadrado
# Calculo de R2
mse =mean_squared_error(y_test,y_test_pred)
rmse_rf =(mean_squared_error(y_test,y_test_pred))**(1/2)
r2 = r2_score(y_test,y_test_pred)
mse1 =mean_squared_error(y_test1,y_test_pred1)
rmse_rf1 =(mean_squared_error(y_test1,y_test_pred1))**(1/2)
r21 = r2_score(y_test1,y_test_pred1)
print("************************************************************")
print("**********       MSE {}:                    {}".format(equipo, mse))
print("**********       ERROR CUADRÁTICO MEDIO {}: {}".format(equipo, rmse_rf))
print("**********       R2 {}:                     {}".format(equipo, r2))
print("**********       MSE {}:                    {}".format(equipo1, mse1))
print("**********       ERROR CUADRÁTICO MEDIO {}: {}".format(equipo1, rmse_rf1))
print("**********       R2 {}:                     {}".format(equipo1, r21))
print("************************************************************")
```