1 Exercici

Agafa un conjunt de dades de tema esportiu que t'agradi i normalitza els atributs categòrics en dummy. Normalitza els atributs numèrics amb StandardScaler

Per fer l'exercici agafem datasets de les estadístiques per lligues de futbol que hi han a [football-data.co.uk (https://www.football-data.co.uk/englandm.php)](https://www.football-data.co.uk/englandm.php).

Extreurem les dades d'un equip en concret de la premier league anglesa desde l'any 2000 fins l'actualitat. Haurem d'importar els datasets de cadascuna de les temporades i concatenar-los en un de sol. Posteriorment transformarem les dades perque ens mostrin la informació des del punt de vista subjectiu de l'equip en qüestió

In [1]:
```
import os
import glob

import math
import pandas as pd
from scipy.stats import shapiro
from sklearn import preprocessing
from sklearn.decomposition import PCA

import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

import datetime
```

executed in 2.48s, finished 22:08:15 2021-05-31

In [2]:
```
#generem una llista amb el paths dels csv a importar que ens hem descarregat a un directori
datasets_path = r"D:\Oscar\FORMACIO\DIGITAL\DATA SCIENCE with Python\Datasets\football stats\Premier league" + os.sep
all_files = glob.glob(datasets_path + "/*.csv")

#importem tots el datasets en un de sol excepte el de la de la temporada actual, que com s'està jugant, la importem d'URL
pd.options.display.max_columns = None
df = pd.concat((pd.read_csv(f) for f in all_files))
df = pd.concat([df, pd.read_csv("https://www.football-data.co.uk/mmz4281/2021/E0.csv")])
df.sample(3)
```

executed in 1.03s, finished 22:08:16 2021-05-31

Out[2]:

	Div	Date	HomeTeam	AwayTeam	FTHG	FTAG	FTR	HTHG	HTAG	HTR	Attendance	Referee	HS	AS	HST	AST
187	E0	22/12/2002	Liverpool	Everton	0.0	0.0	D	0.0	0.0	D	NaN	G Poll	14.0	5.0	8.0	2.0
48	E0	23/10/2020	Aston Villa	Leeds	0.0	3.0	A	0.0	0.0	D	NaN	P Tierney	12.0	27.0	4.0	9.0
183	E0	21/12/2002	Newcastle	Fulham	2.0	0.0	H	1.0	0.0	H	NaN	A Wiley	19.0	6.0	14.0	0.0

In [3]:
```
#comprovant valors nulls veiem que ens surt una fila sencera de NaNs
df[df.HomeTeam.isna()]
```

executed in 172ms, finished 22:08:16 2021-05-31

Out[3]:

	Div	Date	HomeTeam	AwayTeam	FTHG	FTAG	FTR	HTHG	HTAG	HTR	Attendance	Referee	HS	AS	HST	AST	HH
380	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	

In [4]:
```
#l'esborrem doncs és buida
df.dropna(subset=["HomeTeam"], inplace=True)
df.shape
```

executed in 45ms, finished 22:08:16 2021-05-31

Out[4]: (7980, 166)

[informació les columnes (https://www.football-data.co.uk/notes.txt)](https://www.football-data.co.uk/notes.txt)

Columnes que tenim en compte:

- Date = Match Date (dd/mm/yy)
- HomeTeam = Home Team
- AwayTeam = Away Team
- FTHG and HG = Full Time Home Team Goals
- FTAG and AG = Full Time Away Team Goals
- FTR and Res = Full Time Result (H=Home Win, D=Draw, A=Away Win)

Match Statistics (where available)

- HS = Home Team Shots
- AS = Away Team Shots
- HST = Home Team Shots on Target
- AST = Away Team Shots on Target
- HC = Home Team Corners
- AC = Away Team Corners
- HF = Home Team Fouls Committed

- AF = Away Team Fouls Committed
- HY = Home Team Yellow Cards
- AY = Away Team Yellow Cards
- HR = Home Team Red Cards
- AR = Away Team Red Cards

In [5]:
```
#tenim tots aquests equips al dataset
df.HomeTeam.unique()
```
executed in 27ms, finished 22:08:16 2021-05-31

Out[5]:
```
array(['Charlton', 'Chelsea', 'Coventry', 'Derby', 'Leeds', 'Leicester',
       'Liverpool', 'Sunderland', 'Tottenham', 'Man United', 'Arsenal',
       'Bradford', 'Ipswich', 'Middlesbrough', 'Everton', 'Man City',
       'Newcastle', 'Southampton', 'West Ham', 'Aston Villa', 'Bolton',
       'Blackburn', 'Fulham', 'Birmingham', 'West Brom', 'Portsmouth',
       'Wolves', 'Norwich', 'Crystal Palace', 'Wigan', 'Reading',
       'Sheffield United', 'Watford', 'Hull', 'Stoke', 'Burnley',
       'Blackpool', 'QPR', 'Swansea', 'Cardiff', 'Bournemouth',
       'Brighton', 'Huddersfield'], dtype=object)
```

In [6]:
```
#utilitzarem la funció definida en la pràctica d'estadística que donat un equip pel nostre dataset ens retorna un dataframe
#amb la informació subjectiva i unificada per l'equip en concret, i per partit
def df_de_dades_x_team(team, df=df):
    new_cols = ["DATE", "TEAM", "GOLS_FAVOR", "GOLS_CONTRA", "RESULTAT", "XUTS", "XUTS_REBUTS",
                "XUTS_PORTA", "XUTS_PORTA_REBUTS", "CORNERS_LLENÇATS", "CORNERS_REBUTS",
                "FALTES_COMESES", "FALTES_REBUDES", "TARGETES"]

    #partits de local
    old_cols = ["Date", "HomeTeam", "FTHG", "FTAG", "FTR", "HS", "AS",
                "HST", "AST", "HC", "AC", "HF", "AF", "HY"]
    local_team_df = df[(df.HomeTeam == team)]
    rename = dict(zip(old_cols, new_cols))
    local_team_df = local_team_df.rename(columns=rename)[new_cols]
    #Mapegem la columna de RESULTAT considerant que es local
    resultat_dic = {"H": "Win", "D": "Draw", "A": "Lose"}
    local_team_df.RESULTAT = local_team_df.RESULTAT.map(resultat_dic)

    #partits de visitant
    old_cols = ["Date", "AwayTeam", "FTAG", "FTHG", "FTR", "AS", "HS",
                "AST", "HST", "AC", "HC", "AF", "HF", "AY"]
    away_team_df = df[df.AwayTeam == team]
    rename = dict(zip(old_cols, new_cols))
    away_team_df = away_team_df.rename(columns=rename)[new_cols]
    #Mapegem la columna de RESULTAT considerant que es visitant
    resultat_dic = {"H": "Lose", "D": "Draw", "A": "Win"}
    away_team_df.RESULTAT = away_team_df.RESULTAT.map(resultat_dic)

    #afegim una columna booleana per especificicar si juga de local o visitant
    local_team_df["LOCAL"] = True
    away_team_df["LOCAL"] = False

    #finamlment retornem els dos dataframes concatenats
    return local_team_df.append(away_team_df)
```
executed in 28ms, finished 22:08:16 2021-05-31

In [7]:
```
#fem un nou dataframe de cada partit desde el punt de vista subjectiu d'un equip: "Man City"
team = "Man City"
team_data_df = df_de_dades_x_team(team)
team_data_df.sample(3)
```
executed in 77ms, finished 22:08:16 2021-05-31

Out[7]:

	DATE	TEAM	GOLS_FAVOR	GOLS_CONTRA	RESULTAT	XUTS	XUTS_REBUTS	XUTS_PORTA	XUTS_PORTA_REBUTS	CORNERS_LLE
175	26/12/13	Man City	2.0	1.0	Win	20.0	12.0	6.0	5.0	
90	28/11/2020	Man City	5.0	0.0	Win	19.0	9.0	6.0	1.0	
91	26/10/2019	Man City	3.0	0.0	Win	25.0	11.0	9.0	5.0	

1.1 Atributs categòrics en dummy

Com a consideració previa respecte a la transformació numèrica binària de les característiques categòriques podem afirmar que de cadascun dels possibles valors que prengui la característica sempre n'hi haurà un que no ens aportarà informació i podem codificar els possibles valors de cada categoria amb una dimensió menys (és a dir, p.ex. considerant la informació categòrica R:Win, R:Draw y R:Loose, sempre que dos d'ells siguin 0 el tercer serà 1, y sempre que de dos d'ells nhi hagi un que sigui 1, el tercer sera 0). Per aquesta raó, per tal d'evitar redundàncies, i de reduïr les dimensions del dataframe, aplicarem l'argument ***drop_first=True****.*

In [8]:
```
#convertim a numèriques binàries la categòrica de RESULTAT generant noves columnes amb get_dummies
team_data_df = pd.get_dummies(data=team_data_df, columns=["RESULTAT"], prefix="R", prefix_sep=':', drop_first=True)

#passem la columna booleana a numèrica binària
team_data_df.LOCAL = team_data_df.LOCAL.astype("int")

team_data_df.sample(3)
```

executed in 59ms, finished 22:08:16 2021-05-31

Out[8]:

	DATE	TEAM	GOLS_FAVOR	GOLS_CONTRA	XUTS	XUTS_REBUTS	XUTS_PORTA	XUTS_PORTA_REBUTS	CORNERS_LLENÇATS	CC
61	30/09/2017	Man City	1.0	0.0	17.0	4.0	6.0	2.0	8.0	
370	12/05/2019	Man City	4.0	1.0	20.0	6.0	9.0	2.0	6.0	
350	04/05/13	Man City	0.0	0.0	12.0	7.0	8.0	1.0	4.0	

In [9]:
```
#la columna temporal "DATE" porta inplícita informació categòrica que podriem extreure i passar a numèrica com per exemple
#el dia de la setmana (també du dades numèriques com dia, mes i any)
#podem observar que aL dataset hi ha dates en format MM/DD/AA i MM/DD/AAAA (hem de tenir-ho en compte)
def convert_datetime(date):
    if len(date) == 8: return pd.to_datetime(date, format='%d/%m/%y')
    else: return pd.to_datetime(date, format='%d/%m/%Y')

team_data_df.DATE = team_data_df.DATE.apply(convert_datetime)

#generem les noves columnes d'informació de data (agafarem dia setmana de categòrica i any de numèrica)
team_data_df["DIA_SET"] = team_data_df.DATE.dt.day_name()
team_data_df["ANY"] = team_data_df.DATE.dt.year

#passem a columnes numèriques binàries els dies de la setmana
team_data_df = pd.get_dummies(data=team_data_df, columns=["DIA_SET"], prefix="D", prefix_sep=':', drop_first=True)

#esborrem la columna de DATE i de l'equip
team_data_df.drop(columns=["DATE", "TEAM"], inplace=True)
team_data_df.sample(3)
```

executed in 298ms, finished 22:08:17 2021-05-31

Out[9]:

FALTES_REBUDES	TARGETES	LOCAL	R:Lose	R:Win	ANY	D:Monday	D:Saturday	D:Sunday	D:Thursday	D:Tuesday	D:Wednesday
9.0	1.0	1	1	0	2004	0	1	0	0	0	0
13.0	1.0	1	0	1	2019	0	0	1	0	0	0
17.0	4.0	0	1	0	2005	0	1	0	0	0	0

1.2 Normalitza els atributs numèrics amb StandardScaler

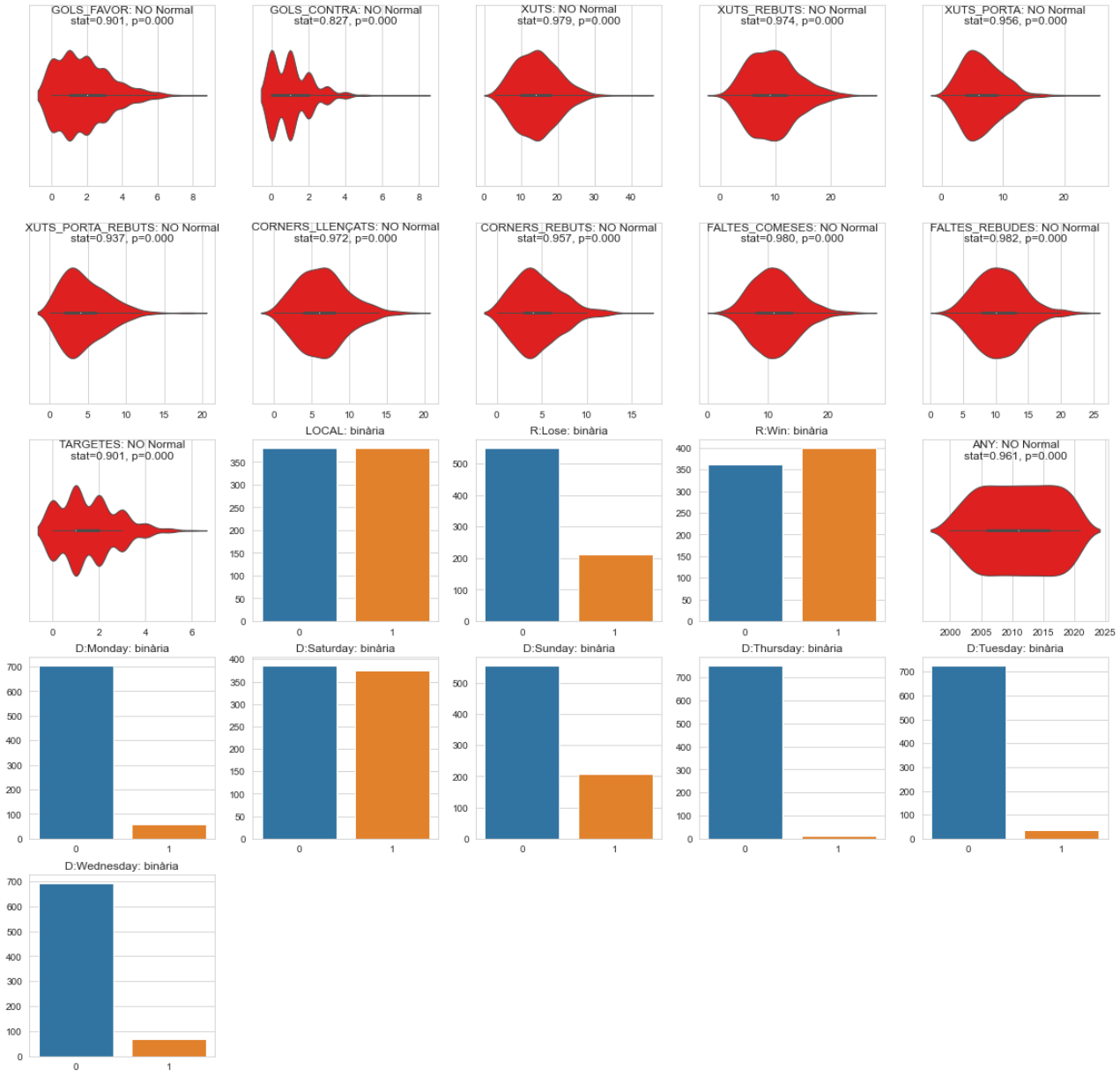
In [10]:

```
#funció per identificar les columnes en binàries y no binaries. De les no binàries dibuixarà les distribucions en violinplot
#i realitzarà, amb shapiro, el test de normalitat (verd les que es distribueixen normalment i vermell les que no),
#i de les primeres, les binàries, dibuixarà el barplot

def plot_distributions(df, cols):
    cols = list(cols)
    dec, num = math.modf(len(cols)/5)
    if dec == 0: rows = int(num)
    else: rows = int(num+1)

    sns.set_style("whitegrid")
    plt.figure(figsize=(20,rows*4))

    binary = {0, 1}
    for i, col in zip(range(len(cols)), cols):
        counts = df[col].value_counts()
        if set(counts.index.to_list()) == binary:
            plt.subplot(rows, 5, i+1)
            sns.barplot(x=counts.index, y=counts.values)
            plt.title(col + ": binària")
            plt.xlabel("")
        else:
            stat, p = shapiro(df[col])
            if p > 0.05:
                title = '{}: Normal\n'.format(col)
                color = "g"
            else:
                title = '{}: NO Normal\n'.format(col)
                color = "r"
            title += 'stat=%.3f, p=%.3f\n' % (stat, p)
            plt.subplot(rows, 5, i+1)
            ax = sns.violinplot(x=col, data=df, color=color, orient='h', linewidth=1 , width=.5)
            if len(set(counts)) == 2:
                title += "Dicotòmica"
                plt.xticks(ticks=counts.index, labels=counts.index.map('{:,.2f}'.format))
            plt.title(title, y=.8)
            plt.xlabel("")
    plt.show()
```

executed in 29ms, finished 22:08:17 2021-05-31

In [11]:
```
#visualitzem les dades numèriques
plot_distributions(team_data_df, team_data_df.columns)
```

executed in 3.44s, finished 22:08:20 2021-05-31



In [12]:
```
#Estandaritzem les característiques amb StandardScaler
standardscaler = preprocessing.StandardScaler()
standarized_data_df = pd.DataFrame(standardscaler.fit_transform(team_data_df), columns=team_data_df.columns)
standarized_data_df.head()
```

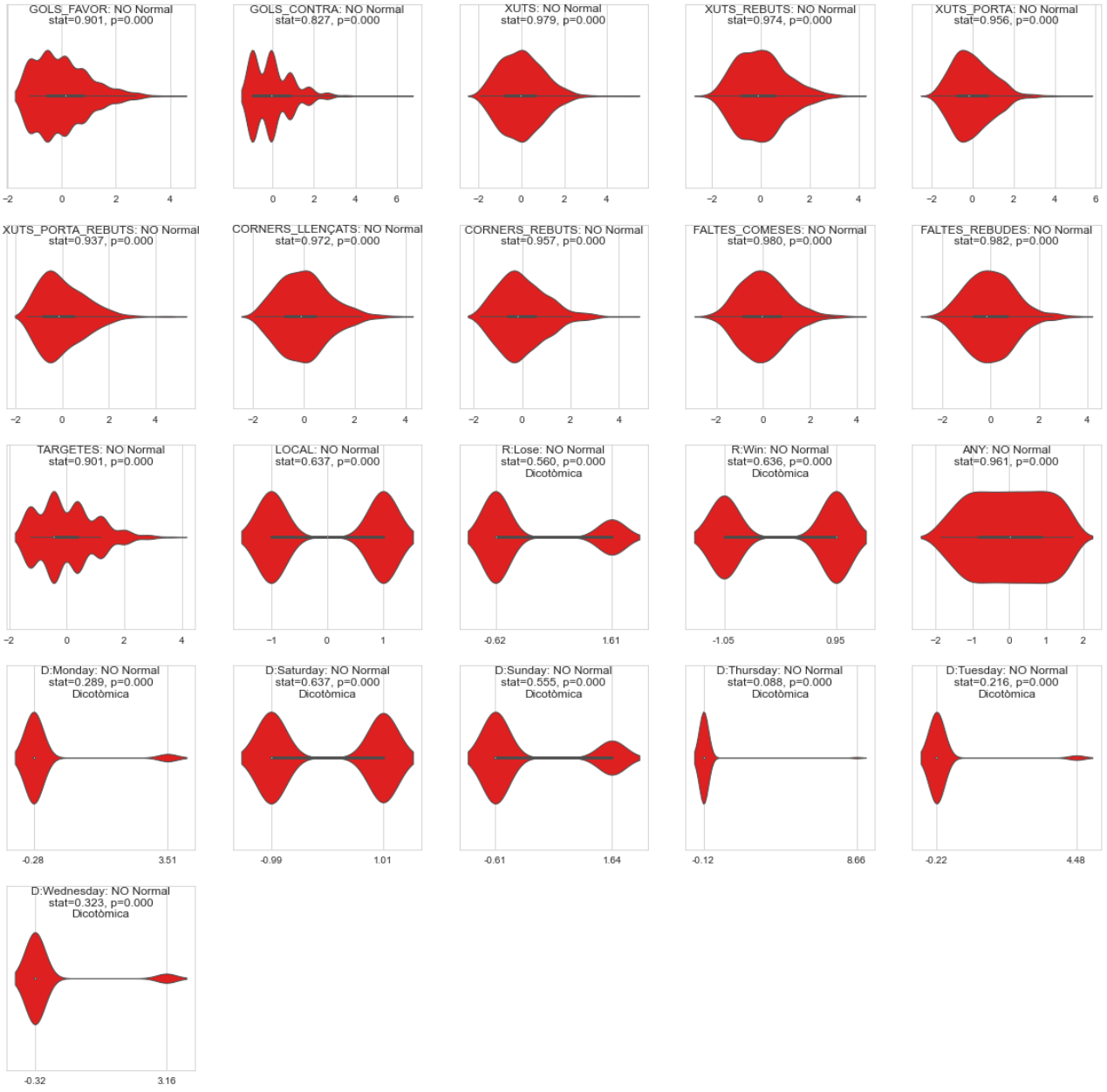
executed in 108ms, finished 22:08:20 2021-05-31

Out[12]:

	GOLS_FAVOR	GOLS_CONTRA	XUTS	XUTS_REBUTS	XUTS_PORTA	XUTS_PORTA_REBUTS	CORNERS_LLENÇATS	CORNERS_REBUTS	F
0	1.439102	0.832061	0.114559	-0.122421	1.048369	-0.170284	0.166923	-0.575332	
1	-0.536306	0.832061	-0.059571	-0.122421	-0.472166	1.157060	-0.432895	0.177254	
2	-0.536306	-0.071319	-0.581959	-1.262612	-0.776273	-0.833956	0.766741	-0.575332	
3	-1.194775	-0.071319	-0.930218	1.245809	0.136048	1.488896	1.066650	1.306133	
4	0.122163	-0.974700	-0.233700	0.105618	1.048369	-0.833956	-0.432895	0.553547	

In [13]:
```
#dibuixem les dades estandaritzades. Podem observar que els seus rangs de valors s'ajusten als d'una distribució normal
#tipificada, i com és una transformació lineal conserven la seva distribució original (totes no normals amb shapiro)
plot_distributions(standarized_data_df, standarized_data_df.columns)
```

executed in 2.57s, finished 22:08:23 2021-05-31



2 Exercici

Continua amb el conjunt de dades de tema esportiu que t'agradi i aplica l'anàlisi de components principals.

In [14]:
```
#com ja tenim les dades estandatitzades (requisit pel correcte funcionament del PCA), utilitzem el dataframe creat
pca = PCA()
pca.fit(standarized_data_df)

#veiem el porcentatge de variança explicada per cada component
pca.explained_variance_ratio_
```

executed in 61ms, finished 22:08:23 2021-05-31

Out[14]:
```
array([0.21150423, 0.09161749, 0.08290947, 0.08107607, 0.06177937,
       0.05876218, 0.05376342, 0.05226594, 0.04928475, 0.04589127,
       0.03749405, 0.03417536, 0.031905  , 0.02518621, 0.02079867,
       0.02071234, 0.01447472, 0.00924178, 0.00912783, 0.00738627,
       0.00064356])
```

In [15]:
```
#influència de les variables en cada component
index = ["PC" + str(i) for i in range(pca.n_components_)]
pca_df = pd.DataFrame(data=pca.components_, columns=standarized_data_df.columns, index=index)
pca_df
```

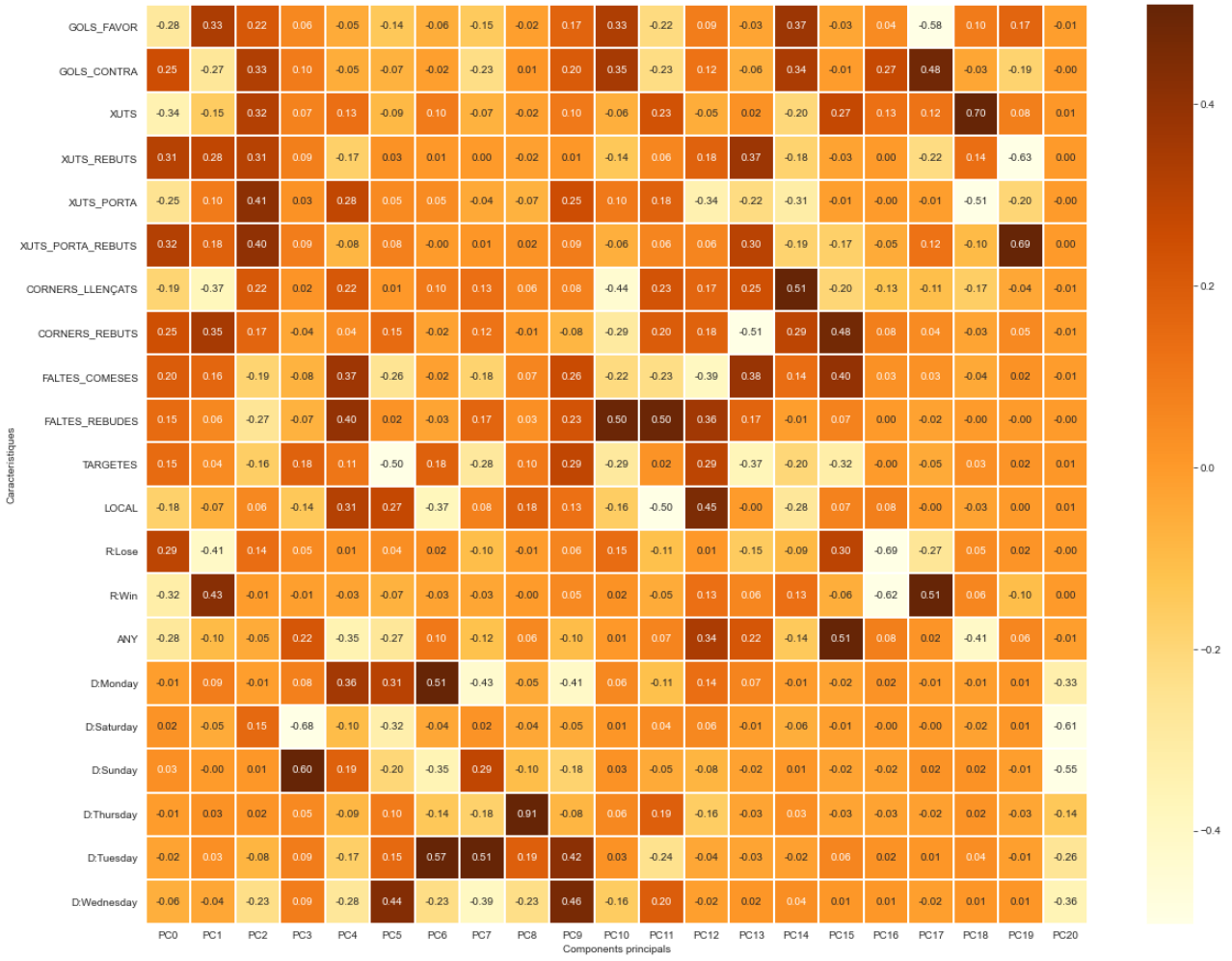
executed in 91ms, finished 22:08:23 2021-05-31

Out[15]:

	GOLS_FAVOR	GOLS_CONTRA	XUTS	XUTS_REBUTS	XUTS_PORTA	XUTS_PORTA_REBUTS	CORNERS_LLENÇATS	CORNERS_REBUTS
PC0	-0.282632	0.251188	-0.340422	0.308577	-0.249604	0.317563	-0.191879	0.245920
PC1	0.334377	-0.272517	-0.154711	0.284877	0.096758	0.181843	-0.374060	0.347850
PC2	0.216920	0.327774	0.316293	0.308383	0.411180	0.404865	0.215987	0.168238
PC3	0.055136	0.096949	0.072161	0.085506	0.026605	0.088344	0.017679	-0.037668
PC4	-0.047681	-0.052825	0.131155	-0.172613	0.284325	-0.080423	0.216380	0.044726
PC5	-0.141253	-0.069495	-0.089561	0.026911	0.050219	0.081769	0.006688	0.149604
PC6	-0.058581	-0.015993	0.096561	0.005248	0.045490	-0.002315	0.101972	-0.021521
PC7	-0.148885	-0.225721	-0.068550	0.000005	-0.043186	0.014943	0.132659	0.119786
PC8	-0.015825	0.012441	-0.017345	-0.018546	-0.068261	0.020116	0.059909	-0.008027
PC9	0.171997	0.204806	0.101011	0.011330	0.249164	0.092595	0.077668	-0.080771
PC10	0.329603	0.350334	-0.063436	-0.142109	0.101977	-0.060273	-0.437974	-0.290978
PC11	-0.215679	-0.225043	0.234448	0.059319	0.178246	0.063584	0.231083	0.199861
PC12	0.089895	0.124129	-0.053221	0.179282	-0.339734	0.063601	0.166804	0.178929
PC13	-0.031634	-0.059935	0.023766	0.373845	-0.221107	0.301450	0.245213	-0.510832
PC14	0.374388	0.337430	-0.204662	-0.183216	-0.309956	-0.185099	0.512938	0.294947
PC15	-0.032411	-0.006529	0.273853	-0.027502	-0.013088	-0.173051	-0.195894	0.475006
PC16	0.043662	0.271674	0.128262	0.002607	-0.002280	-0.049999	-0.126766	0.083961
PC17	-0.580029	0.478929	0.119593	-0.220671	-0.008389	0.123738	-0.110049	0.043985
PC18	0.095353	-0.033653	0.698452	0.136771	-0.513341	-0.097632	-0.165506	-0.029771
PC19	0.166103	-0.185101	0.084925	-0.626773	-0.195426	0.691272	-0.038735	0.052174
PC20	-0.009129	-0.004945	0.010410	0.001203	-0.003356	0.002014	-0.009468	-0.014974

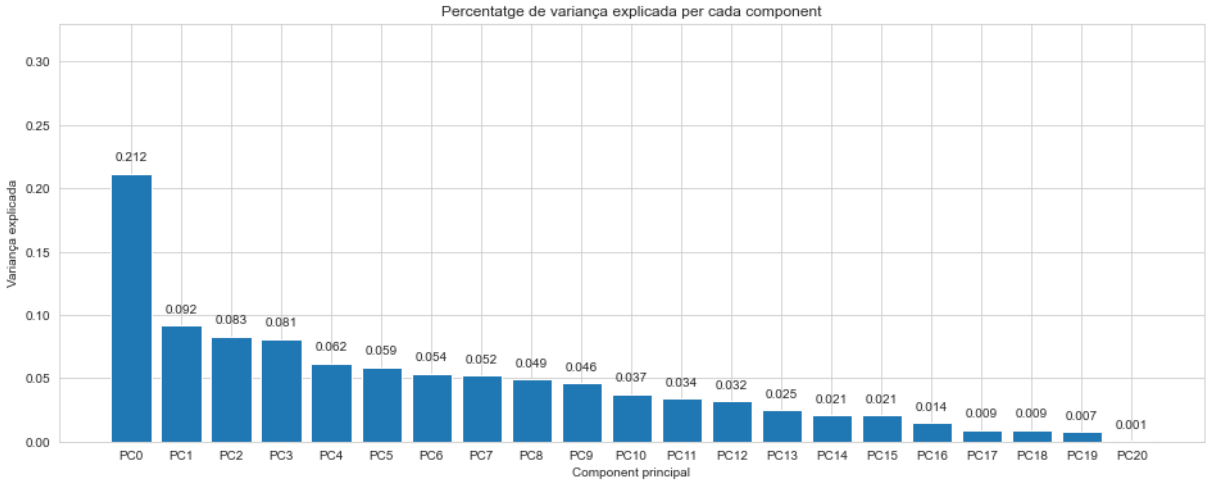
In [16]:
```
#Podem visualitzar La influència de les variables en cada component amb un heatmap
plt.figure(figsize=(20,16))
sns.heatmap(pca_df.T, annot=True, annot_kws={"size": 10}, cmap="YlOrBr", fmt='.2f', robust=True, linewidths=1, linecolor="W")
plt.xlabel("Components principals")
plt.ylabel("Característiques")
plt.show()
```

executed in 5.49s, finished 22:08:28 2021-05-31



In [17]:
```
#Dibuixem amb un gràfic de barres aquest percentatge de variança explicada per cada component
plt.figure(figsize=(16, 6))
plt.bar(x=np.arange(pca.n_components_), height=pca.explained_variance_ratio_)

for x, y in zip(np.arange(len(standarized_data_df.columns)), pca.explained_variance_ratio_):
    plt.annotate(round(y, 3), (x,y), textcoords="offset points", xytext=(0,10), ha='center')

plt.xticks(np.arange(pca.n_components_), labels=pca_df.index)
plt.ylim(0, .33)
plt.title('Percentatge de variança explicada per cada component')
plt.xlabel('Component principal')
plt.ylabel('Variança explicada')
plt.show()
```

executed in 715ms, finished 22:08:29 2021-05-31



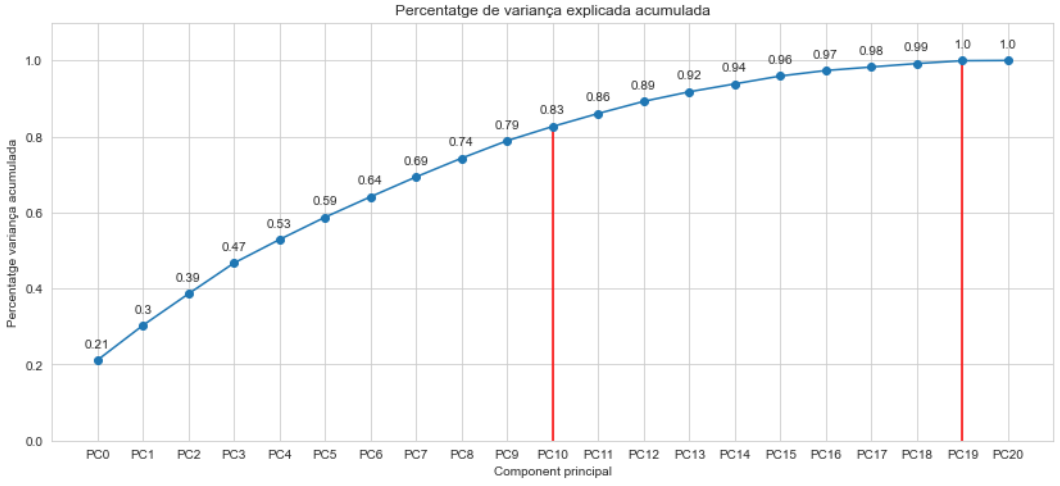
In [18]:
```
#Observem el percentatge de variança explicada acumulada
pca.explained_variance_ratio_.cumsum()
```

executed in 13ms, finished 22:08:29 2021-05-31

Out[18]:
```
array([0.21150423, 0.30312172, 0.38603119, 0.46710727, 0.52888663,
       0.58764881, 0.64141223, 0.69367817, 0.74296292, 0.78885419,
       0.82634824, 0.8605236 , 0.89242861, 0.91761481, 0.93841349,
       0.95912582, 0.97360055, 0.98284233, 0.99197016, 0.99935644,
       1.        ])
```

In [19]:
```
#Podem dibuixar la acumulació
varianca_acum = pca.explained_variance_ratio_.cumsum()

plt.figure(figsize=(14, 6))
plt.plot(np.arange(len(standarized_data_df.columns)), varianca_acum, marker='o')

for x, y in zip(np.arange(len(standarized_data_df.columns)), varianca_acum):
    label = round(y, 2)
    plt.annotate(label, (x,y), textcoords="offset points", xytext=(0,10), ha='center')

plt.ylim(0, 1.1)
plt.xticks(np.arange(pca.n_components_), labels=pca_df.index)
plt.vlines(x=[10, 19], ymin=0, ymax=[varianca_acum[10], 1], color="r")

plt.title('Percentatge de variança explicada acumulada')
plt.xlabel('Component principal')
plt.ylabel('Percentatge variança acumulada')
plt.show()
```

executed in 666ms, finished 22:08:30 2021-05-31



Podem concloure que aplicant el metode d'extracció dels components principals al nostre conjunt de dades:

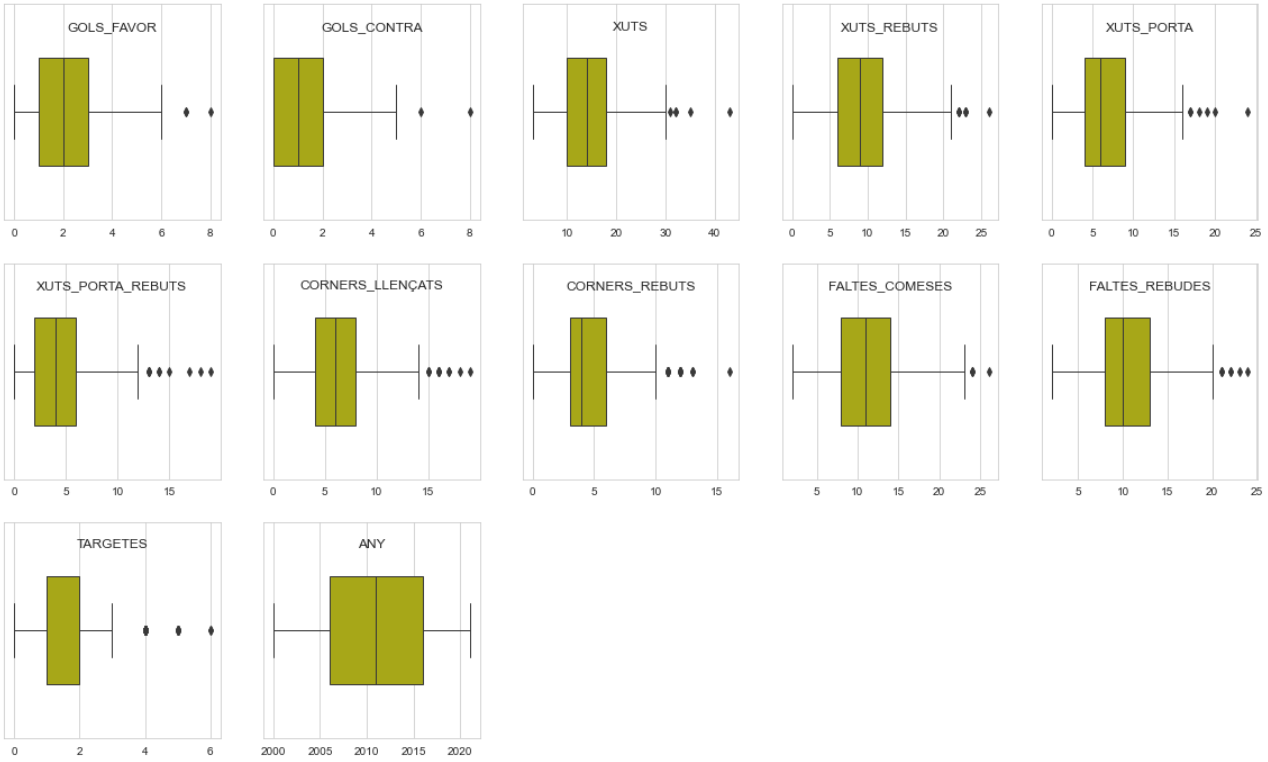
- *podem reduïr el nou dataframe de PCs en una dimensió respecte al de l'origen de dades sense perdre cap informació.*
- *podem reduïr a quasi la meitat de dimensions (de 21 a 11) conservant el 83% de la informació*

3 Exercici

Continua amb el conjunt de dades de tema esportiu que t'agradi i normalitza les dades tenint en compte els outliers.

In [20]:
```
#definim una funció per dibuixar les distribucions en boxplot de les columnes especificades d'un dataframe
def boxplots(df, cols, color="y"):
    cols = list(cols)
    dec, num = math.modf(len(cols)/5)
    if dec == 0: rows = int(num)
    else: rows = int(num+1)

    sns.set_style("whitegrid")
    plt.figure(figsize=(20,rows*4))
    for i, col in zip(range(len(cols)), cols):
        plt.subplot(rows, 5, i+1)
        ax = sns.boxplot(x=col, data=df, color=color, orient='h', linewidth=1 , width=.5)
        plt.title(col, y=.85)
        plt.xlabel("")
    plt.show()

#separem les columnes en binaries i no binàries, en dues llistes
binary = {0, 1}
cat_cols, num_cols = [], []

for col in team_data_df.columns:
    if set(team_data_df[col].value_counts().index.to_list()) != binary:
        num_cols.append(col)
    else: cat_cols.append(col)

#per tractar els outliers ens quedarem amb les columnes numèriques amb dades no binàries.
#dibuixem les distribucions de les nostres dades per observar, amb els límits del boxplot, els valors atípics (en punts)
boxplots(team_data_df, num_cols)
```

executed in 1.59s, finished 22:08:31 2021-05-31



3.1 Mètode ajustant ouliers a màxims i mínims i normalització amb MinMaxScaler

Tenint en compte, per definició, q els valors atípics son aquells que es trobin per sobre de $q3 + 1.5 * IQR$ o per sota de $q1 - 1.5 * IQR$ depen de la nostra desició si els prenem com errors (i els eliminem), o si els considerem com a casos excepcionals (i els establim a un màxim/mínim perque no ens modifiquin amb un factor tant elevat les tendències centrals).

Essent, el nostre cas, dades extretes d'una web oficial i amb poca possiblitat d'error, els tractarem com el segon cas i els canviarem de valor a un màxim/mínim no atipic.

In [21]:
```
#definim un procés per crear un nou dataframe sense outliers. Els establirà als valor màxims i mínims dels límits
#no considerats com a outliers, quan els trobi
def minmax_outliers(series):
    riq = series.quantile(q=.75) - series.quantile(q=.25)
    return series.quantile(q=.25) - 1.5*riq, series.quantile(q=.75) + 1.5*riq

def fix_outliers(value, minim, maxim):
    if value < minim: return minim
    elif value > maxim: return maxim
    else: return value

team_data_wo_out_df = pd.DataFrame()
for col in num_cols:
    minim, maxim = minmax_outliers(team_data_df[col])
    team_data_wo_out_df[col] = team_data_df[col].apply(fix_outliers, args=(minim, maxim))
team_data_wo_out_df.head()
```

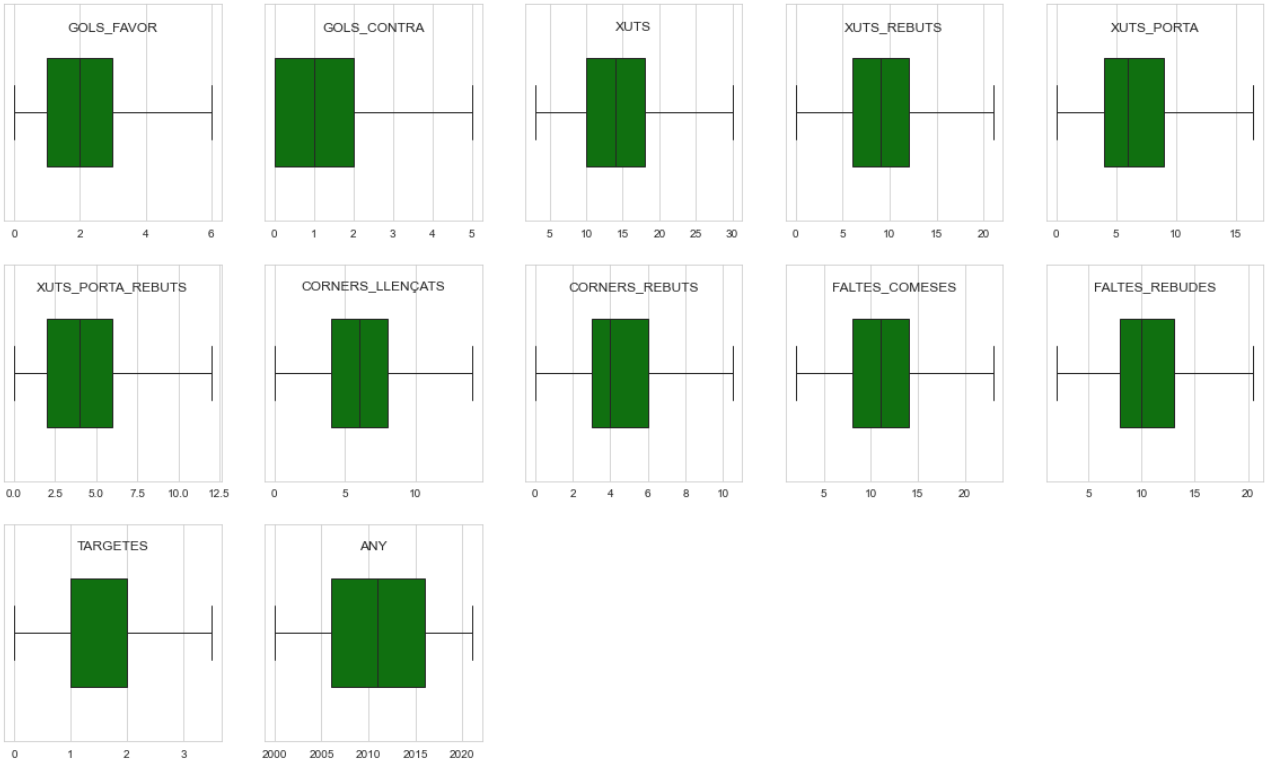
executed in 125ms, finished 22:08:31 2021-05-31

Out[21]:

	GOLS_FAVOR	GOLS_CONTRA	XUTS	XUTS_REBUTS	XUTS_PORTA	XUTS_PORTA_REBUTS	CORNERS_LLENÇATS	CORNERS_REBUTS	FA
15	4.0	2.0	15.0	9.0	10.0	4.0	7.0	3.0	
23	1.0	2.0	14.0	9.0	5.0	8.0	5.0	5.0	
57	1.0	1.0	11.0	4.0	4.0	2.0	9.0	3.0	
73	0.0	1.0	9.0	15.0	7.0	9.0	10.0	8.0	
85	2.0	0.0	13.0	10.0	10.0	2.0	5.0	6.0	

In [22]:
```
#dibuixem ara les noves distribucions de les nostres dades, amb els outliers modificats
boxplots(team_data_wo_out_df, num_cols, "g")
```

executed in 1.46s, finished 22:08:33 2021-05-31



In [23]:
```
#podem observar, comparant les mitjanes, abans i despres de tractar els outliers que com tots els ouliers eren a la part
#positiva, i no eren massa extrems, han baixat totes les mitjanes una mica exepte ANY, que no en tenia cap
pd.concat([team_data_df[num_cols].mean(), team_data_wo_out_df.mean()], axis=1, keys=["mean original","mean sense outliers"])
```

executed in 45ms, finished 22:08:33 2021-05-31

Out[23]:

	mean original	mean sense outliers
GOLS_FAVOR	1.814474	1.809211
GOLS_CONTRA	1.078947	1.073684
XUTS	14.342105	14.311842
XUTS_REBUTS	9.536842	9.518421
XUTS_PORTA	6.552632	6.531579
XUTS_PORTA_REBUTS	4.513158	4.475000
CORNERS_LLENÇATS	6.443421	6.406579
CORNERS_REBUTS	4.528947	4.487500
FALTES_COMESES	11.242105	11.234211
FALTES_REBUDES	10.610526	10.596711
TARGETES	1.551316	1.498026
ANY	2010.936842	2010.936842

In [24]:
```
#concatenem les columnes numèriques binàries, que hem separat, per no haver-les de tractar
team_data_wo_out_df = pd.concat([team_data_wo_out_df, team_data_df[cat_cols]], axis=1)

#per continuar amb els requisits de l'excercici, normalitzem els valors. Com no s'especifica quin tipus de normalització
#i, segons el contexte se n'aplica una o altre, en aquest cas com no hi ha números negatius aplicarem la MinMax scaler per un
#rang d'entrada d'entre de 0 a 1 (default)
minmaxscaler = preprocessing.MinMaxScaler()
normalized_data_df = pd.DataFrame(minmaxscaler.fit_transform(team_data_wo_out_df), columns=team_data_wo_out_df.columns)
normalized_data_df.head()
```

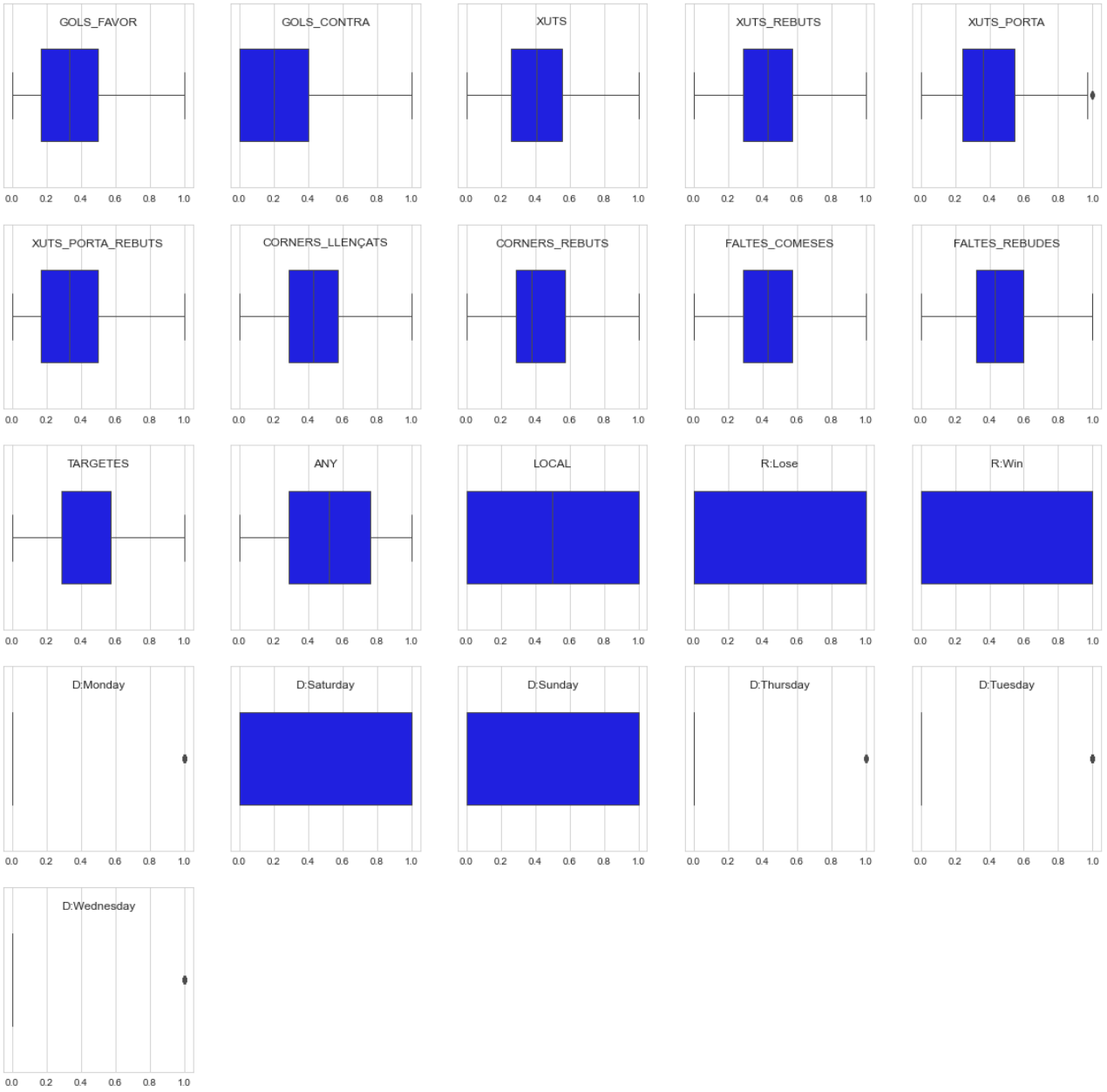
executed in 90ms, finished 22:08:33 2021-05-31

Out[24]:

	GOLS_FAVOR	GOLS_CONTRA	XUTS	XUTS_REBUTS	XUTS_PORTA	XUTS_PORTA_REBUTS	CORNERS_LLENÇATS	CORNERS_REBUTS	F
0	0.666667	0.4	0.444444	0.428571	0.606061	0.333333	0.500000	0.285714	
1	0.166667	0.4	0.407407	0.428571	0.303030	0.666667	0.357143	0.476190	
2	0.166667	0.2	0.296296	0.190476	0.242424	0.166667	0.642857	0.285714	
3	0.000000	0.2	0.222222	0.714286	0.424242	0.750000	0.714286	0.761905	
4	0.333333	0.0	0.370370	0.476190	0.606061	0.166667	0.357143	0.571429	

In [25]:
```
#si dibuixem ara els boxplots, amb els outliers tractats i amb MinMaxScaler. Veiem els rangs de dades de 0 a 1
boxplots(normalized_data_df, normalized_data_df.columns, "b")
```

executed in 2.69s, finished 22:08:36 2021-05-31



3.2 Mètode de escalament amb RobustScaler

Robust Scaler resta la mediana i divideix entre el rang interquantil (IQR) a cada valor i ***s'utilitza quan tenim dades atípiques i no volem que tinguin molta influència****.*

In [26]:
```
#primer creem un procés per generar la normalització segons la definició i després ho farem amb RobustScaler d'sklearn
def rs_value(value, col, df):
    med, riq = df.loc[col, "MED"], df.loc[col, "RIQ"]
    return (value - med) / riq

def robust_scaler(df):
    riq_s = (df.quantile(q=.75) - df.quantile(q=.25)).rename("RIQ")
    med_riq_df = pd.concat([df.median().rename("MED"), riq_s], axis=1)

    rs_df = pd.DataFrame()
    for col in df.columns: rs_df = pd.concat([rs_df, df[col].apply(rs_value, args=(col, med_riq_df))], axis=1)
    return rs_df

normalized_data_df = robust_scaler(team_data_df)
normalized_data_df.head(3)
```

executed in 698ms, finished 22:08:36 2021-05-31

Out[26]:

	GOLS_FAVOR	GOLS_CONTRA	XUTS	XUTS_REBUTS	XUTS_PORTA	XUTS_PORTA_REBUTS	CORNERS_LLENÇATS	CORNERS_REBUTS	FA
15	1.0	0.5	0.125	0.000000	0.8	0.0	0.25	-0.333333	
23	-0.5	0.5	0.000	0.000000	-0.2	1.0	-0.25	0.333333	
57	-0.5	0.0	-0.375	-0.833333	-0.4	-0.5	0.75	-0.333333	

In [27]:
```
#Les columnes numèriques contínues ens ho fa correctament, pero a les binàries veiem rangs interquantils de 0,
#el que ens ocasiona divisions per 0
def robust_scaler(df):
    riq_s = (df.quantile(q=.75) - df.quantile(q=.25)).rename("riq")
    return pd.concat([df.median().rename("mediana"), riq_s], axis=1)
robust_scaler(team_data_df).T.iloc[:,12:]
```

executed in 45ms, finished 22:08:37 2021-05-31

Out[27]:

	R:Lose	R:Win	ANY	D:Monday	D:Saturday	D:Sunday	D:Thursday	D:Tuesday	D:Wednesday
mediana	0.0	1.0	2011.0	0.0	0.0	0.0	0.0	0.0	0.0
riq	1.0	1.0	10.0	0.0	1.0	1.0	0.0	0.0	0.0

In [28]:
```
#veiem que a les columnes binàries ens donen valors d'infinit, producte de la divisió de rangs interquantils de 0 on predomina
#el valor zero
pd.concat([normalized_data_df.min().rename("mínim"), normalized_data_df.max().rename("màxim")], axis=1).T.iloc[:,12:]
```

executed in 45ms, finished 22:08:37 2021-05-31

Out[28]:

	R:Lose	R:Win	ANY	D:Monday	D:Saturday	D:Sunday	D:Thursday	D:Tuesday	D:Wednesday
mínim	0.0	-1.0	-1.1	inf	0.0	0.0	inf	inf	inf
màxim	1.0	0.0	1.0	inf	1.0	1.0	inf	inf	inf

In [29]:
```
#si ho fem amb RobustScaler d'sklearn ens surten el mateixos resultats per les columnes de dades numèriques contínues,
#però les dades d'informació categòrica amb valors binaris ens les tracta difererent doncs hi veiem infinits
robustescaler = preprocessing.RobustScaler()
normalized_data_df = pd.DataFrame(robustescaler.fit_transform(team_data_df), columns=team_data_df.columns)
normalized_data_df.head(3)
```

executed in 60ms, finished 22:08:37 2021-05-31

Out[29]:

	GOLS_FAVOR	GOLS_CONTRA	XUTS	XUTS_REBUTS	XUTS_PORTA	XUTS_PORTA_REBUTS	CORNERS_LLENÇATS	CORNERS_REBUTS	FA
0	1.0	0.5	0.125	0.000000	0.8	0.0	0.25	-0.333333	
1	-0.5	0.5	0.000	0.000000	-0.2	1.0	-0.25	0.333333	
2	-0.5	0.0	-0.375	-0.833333	-0.4	-0.5	0.75	-0.333333	

In [30]:
```
#amb sklearn en aquestes columnes binàries supera d'alguna manera aquestes divisions per 0 doncs sens normalitzen
pd.concat([normalized_data_df.min().rename("mínim"), normalized_data_df.max().rename("màxim")], axis=1).T.iloc[:,12:]
```

executed in 43ms, finished 22:08:37 2021-05-31

Out[30]:

	R:Lose	R:Win	ANY	D:Monday	D:Saturday	D:Sunday	D:Thursday	D:Tuesday	D:Wednesday
mínim	0.0	-1.0	-1.1	0.0	0.0	0.0	0.0	0.0	0.0
màxim	1.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

In [31]:
```
#observem la normalització de les columnes binàries
cols = ["R:Win", "R:Lose"]
compare = []
for col in cat_cols:
    compare.append(pd.concat([team_data_df[col].value_counts(), normalized_data_df[col].value_counts()],
              axis=1, keys=[col+":abans", col+":despres"]))
pd.concat(compare, axis=1)
```

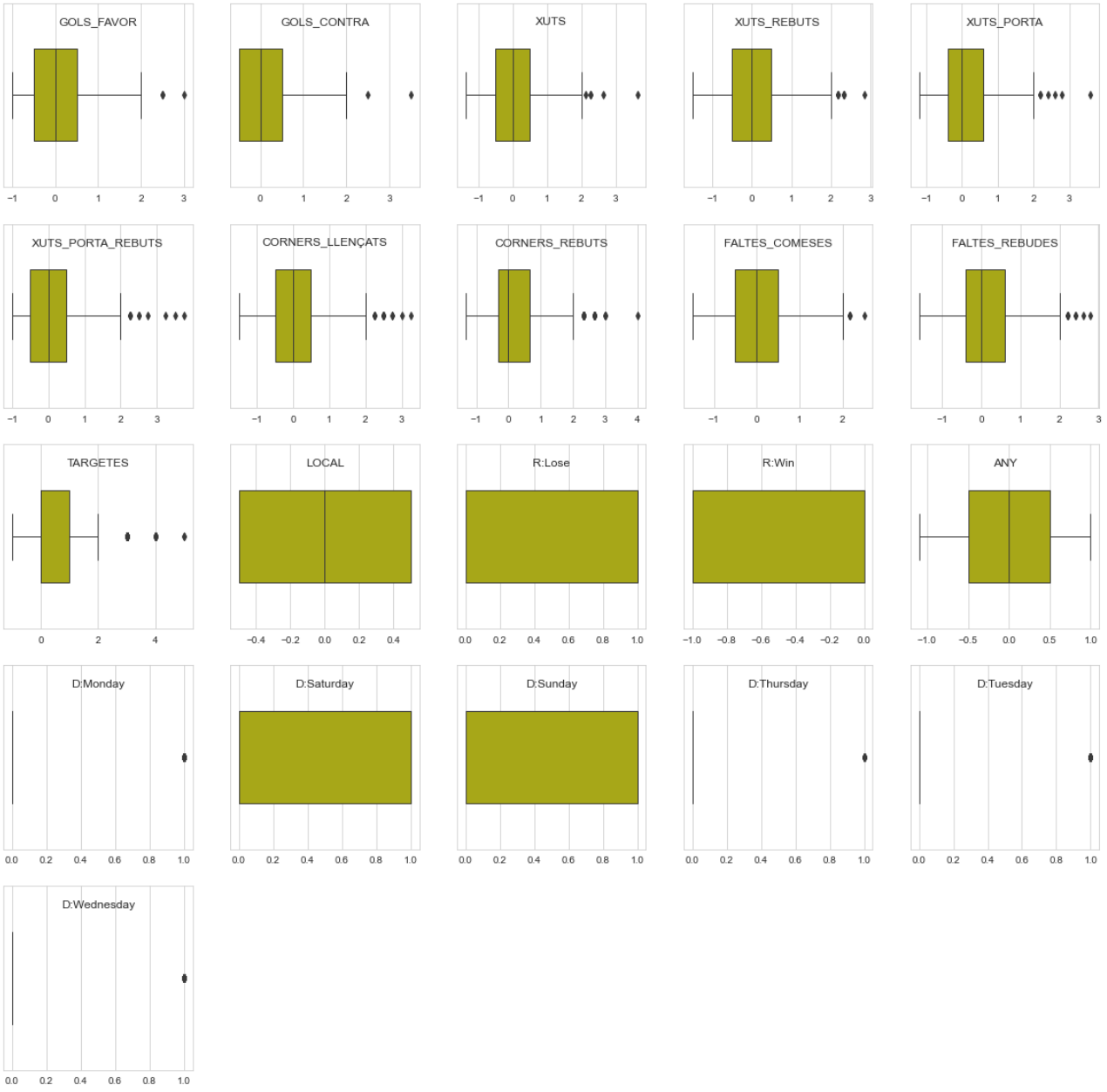
executed in 108ms, finished 22:08:37 2021-05-31

Out[31]:

	LOCAL:abans	LOCAL:despres	R:Lose:abans	R:Lose:despres	R:Win:abans	R:Win:despres	D:Monday:abans	D:Monday:despres	D:Saturda
-1.0	NaN	NaN	NaN	NaN	NaN	361.0	NaN	NaN	
-0.5	NaN	380.0	NaN	NaN	NaN	NaN	NaN	NaN	
0.0	380.0	NaN	549.0	549.0	361.0	399.0	703.0	703.0	
0.5	NaN	380.0	NaN	NaN	NaN	NaN	NaN	NaN	
1.0	380.0	NaN	211.0	211.0	399.0	NaN	57.0	57.0	

In [32]:
```
#dibuixem les distribucions de les nostres dades despres d'aplicar el robust scaler
boxplots(normalized_data_df, normalized_data_df.columns)
```

executed in 2.74s, finished 22:08:40 2021-05-31



Concluïm que amb normalització **robust scaler** (aconsellat amb valors atípics) hi seguim tenint els valors atípics al ser transformacions de dades lineals, tan sols hi disminuïm el seu pes. Si desitgem retallar aquests valors atípics necessitem transformacions no lineals.