Annex

A continuació tenim una explicació més detallada de com funciona el codi desenvolupat en aquest treball.

Annex 1

(aproximadament 165),per tant un partit cada dos dies de mitjana. Això fa que tots els equips més de 10 o 15 vegades a la temporada juguin dos partits en 2 dies o 3 en 4 dies. El cansament que això produeix pot influir de forma important en el resultat del partit.

Per tant el que volem calcular és quants dies de descans ha tingut tant l'equip local i visitant abans d'aquell partit. Per exemple si l'equip local va jugar el dia abans el valor serà 1, si va jugar fa dos dies el paràmetre serà 2 i així succesivament.

Per tal de obtenir aquest nombre de dies de descans de cada equip en cada partit crearem el següent diccionari anomenat last_game. També crearem dos arrays buits on emmagtzamarem els dies de descans tant per l'equip local com per l'equip visitant.

El diccionari tindrà com a clau el nom de l'equip de l'NBA en questió i com a valor associat a la clau un string que contindrà la data de l'últim partit que ha disputat cada equip.

La funció transcorre de la següent manera:

1. Creem un loop amb for i in range( ) de la longitud del dataframe data d'aquesta manera analitzem cada fila que representa un partit de la temporada.
2. Inicialitzem el diccionari per cada equip si no ha jugat (fins que tots els equips no hagin jugat el seu primer partit el diccionari no estarà complert), també assignem un valor al primer partit de cada equip en aquest cas posarem 3 dies de descans per cada equip ja que és el descans que tenen dels partits de pretemporada al primer partit de la temporada.
3. Un cop inicilitzats tots els diccionaris creem 3 variables, next_match que conté la data del partit que volem calcular el descans, last_match_visitor que conté la data de l'últim partit disputat per l'equip visitant, i last_match_local que conté la data de l'últim partit disputat per l'equip local.
4. Finalment comparem les dues dates de next_match, last_match_visitor i next_match, last_match_local. Guardarem la diferència en dies entre els partits en les arrays de local i visitor que hem creat
5. Actualitzem el valor de last match per els dos equips que disputen el partit d'aquesta manera modifiquem el diccionari last_match.
6. La funció retorna el valor de les dues arrays i també el diccionari que conté els últims partits disputats per cada equip

Finalment restem les dues arrays per tal de obtenir un tercer array que ens doni la diferencia entre el nombre de dies descansats dels dos equips.

```
def back_to_back(data,dim):
    last_game={}
    last_visitor=[]
    last_local=[]
    for i in range(dim):
        #Inicialitzem diccionaris i arrays
        if data['Visitor'][i] not in last_game:
            last_game[data['Visitor'][i]]=data['Date'][i]
            last_visitor=np.append(last_visitor,3)
        if data['Home'][i] not in last_game:
            last_game[data['Home'][i]]=data['Date'][i]
            last_local=np.append(last_local,3)

        else:
            next_match=data['Date'][i]

            last_match_visitor=last_game[data['Visitor'][i]]
```

```
            last_match_local=last_game[data['Home'][i]]
            last_visitor=np.append(last_visitor,int(str(next_match-last_match_visitor)[0]))
            last_local=np.append(last_local,int(str(next_match-last_match_local)[0]))


            last_game[data['Visitor'][i]]=data['Date'][i]
            last_game[data['Home'][i]]=data['Date'][i]


    return last_game,last_local,last_visitor

dic_last_game,b2b_local,b2b_visitor=back_to_back(data,len(data))

b2b=[]
for i in range(len(b2b_local)):
    b2b=np.append(b2b,"{0:.2f}".format(float(b2b_local[i])-float(b2b_visitor[i])))

data['Back to back']=b2b
data
```

	Date	Visitor	PTSV	Home	PTSH	Overtime	Attend.	Back to back
0	2019-10-22	New Orleans Pelicans	122	Toronto Raptors	130	1	20,787	0.00
1	2019-10-22	Los Angeles Lakers	102	Los Angeles Clippers	112	0	19,068	0.00
2	2019-10-23	Chicago Bulls	125	Charlotte Hornets	126	0	15,424	0.00
3	2019-10-23	Detroit Pistons	119	Indiana Pacers	110	0	17,923	0.00
4	2019-10-23	Cleveland Cavaliers	85	Orlando Magic	94	0	18,846	0.00
...	...	...	...	...	...	...	...	...
966	2020-03-10	Brooklyn Nets	104	Los Angeles Lakers	102	0	18,997	0.00
967	2020-03-11	Detroit Pistons	106	Philadelphia 76ers	124	0	20,172	1.00
968	2020-03-11	New York Knicks	136	Atlanta Hawks	131	1	15,393	1.00
969	2020-03-11	Charlotte Hornets	109	Miami Heat	98	0	19,6	1.00
970	2020-03-11	Denver Nuggets	97	Dallas Mavericks	113	0	20,302	-1.00

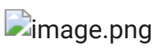
971 rows × 8 columns

Parametritzar la divisió en la que juga cada un dels 30 equips

La següent funció ens permetrà extreure un paràmetre que inidicarà la divisió on juga cada equip.

Primer de tot cal inidicar què són les divisions. Les divisions són grups de 5 equips en els que es subdivideix la lliga, per tant hi ha 6 divisions de 5 equips cada una. Aquestes divisions es fan per proximitat geogràfica ja que és un país enorme. Per evitar fer tants viatges els equips de cada divisió juguen més partits entre ells que amb els altres equips de la lliga. Cada equip juga 5 partits contra els equips de la mateixa divisió 4 contra els de la mateixa conferència i 2 contra els de la altre conferència (les conferències les explicaré a la següent funció).

En la següent imatge es pot veure clarament com funcionen les divisions.



Per poder realitzar aquesta classificació s'ha creat un diccionari que conté com a clau el nom de la divisió i com a valor associat el nom de els equips que hi pertanyen.

Divisions={'Atlantic Division':['Toronto Raptors', 'Boston Celtics', 'Philadelphia 76ers', 'Brooklyn Nets', 'New York Knicks'],'Central Division': ['Milwaukee Bucks', 'Indiana Pacers', 'Chicago Bulls','Detroit Pistons', 'Cleveland Cavaliers'],'SouthEast Division':['Miami Heat', 'Orlando Magic','Washington Wizards','Charlotte Hornets','Atlanta Hawks'],'NorWest Division':['Denver Nuggets','Utah Jazz','Oklahoma City Thunder','Portland Trail Blazers', 'Minnesota Timberwolves'],'Pacific Division':['Los Angeles Lakers','Los Angeles Clippers','Sacramento Kings','Phoenix Suns','Golden State Warriors'],'SouthWest Division':['Houston Rockets','Dallas Mavericks','Memphis Grizzlies','New Orleans Pelicans','San Antonio Spurs']}

```
Div={'Toronto Raptors':1, 'Boston Celtics':1, 'Philadelphia 76ers':1, 'Brooklyn Nets':1, 'New York Knicks':1,
           'Milwaukee Bucks':2, 'Indiana Pacers':2, 'Chicago Bulls':2,'Detroit Pistons':2, 'Cleveland Cavaliers':2,
           'Miami Heat':3, 'Orlando Magic':3,'Washington Wizards':3,'Charlotte Hornets':3,'Atlanta Hawks':3,
           'Denver Nuggets':4,'Utah Jazz':4,'Oklahoma City Thunder':4,'Portland Trail Blazers':4, 'Minnesota Timberwolves':4,
           'Los Angeles Lakers':5,'Los Angeles Clippers':5,'Sacramento Kings':5,'Phoenix Suns':5,'Golden State Warriors':5,
           'Houston Rockets':6,'Dallas Mavericks':6,'Memphis Grizzlies':6,'New Orleans Pelicans':6,'San Antonio Spurs':6}


data['Div_Visitor'] = data.apply(lambda row: Div[row.Visitor] ,axis=1)
data['Div_Home'] = data.apply(lambda row: Div[row.Home] , axis=1)

data['Division'] = data.apply(lambda row: row.Div_Home-row.Div_Visitor, axis=1)
```

```
data
```

	Date	Visitor	PTSV	Home	PTSH	Overtime	Attend.	Back to back	Div_Visitor	Div_Home	Division
0	2019-10-22	New Orleans Pelicans	122	Toronto Raptors	130	1	20,787	0.00	6	1	-5
1	2019-10-22	Los Angeles Lakers	102	Los Angeles Clippers	112	0	19,068	0.00	5	5	0
2	2019-10-23	Chicago Bulls	125	Charlotte Hornets	126	0	15,424	0.00	2	3	1
3	2019-10-23	Detroit Pistons	119	Indiana Pacers	110	0	17,923	0.00	2	2	0
4	2019-10-23	Cleveland Cavaliers	85	Orlando Magic	94	0	18,846	0.00	2	3	1
...	...	...	...	...	...	...	...	...	...	...	...
966	2020-03-10	Brooklyn Nets	104	Los Angeles Lakers	102	0	18,997	0.00	1	5	4
967	2020-03-11	Detroit Pistons	106	Philadelphia 76ers	124	0	20,172	1.00	2	1	-1
968	2020-03-11	New York Knicks	136	Atlanta Hawks	131	1	15,393	1.00	1	3	2
969	2020-03-11	Charlotte Hornets	109	Miami Heat	98	0	19,6	1.00	3	3	0
970	2020-03-11	Denver Nuggets	97	Dallas Mavericks	113	0	20,302	-1.00	4	6	2

971 rows × 11 columns

Volem classificar els equips en funció de si juguen a la conferència Est o Oest

La següent funció ens permetrà extreure un paràmetre que inidicarà si cada un dels equips juga a la conferència Est o Oest.

Primer de tot cal inidicar què són les conferències. La NBA està divida en dos conferències formades per 15 equips cada una, per tant hi ha 2 conferències de 15 equips cada una. Aquestes conferències es fan per proximitat geogràfica ja que Estats Units és un país enorme. Per evitar fer tants viatges llargs els equips de cada conferència juguen més partits entre ells que amb els altres equips de la lliga. Cada equip juga 4 partits contra els de la mateixa conferència i 2 contra els de la altre conferència.

Per poder realitzar aquesta classificació s'ha creat un diccionari que conté com a clau el nom de la conferència i com a valor associat el nom de els equips que hi pertanyen.

```
Conferences={'Eastern Conference':['Toronto Raptors', 'Boston Celtics', 'Philadelphia 76ers', 'Brooklyn Nets', 'New York Knicks','Milwaukee Bucks', 'Indiana Pacers', 'Chicago Bulls
```

```
data['Conference_Visitor'] = data.apply(lambda row: 1 if row.Visitor in Conferences['Eastern Conference'] else 0, axis=1)
data['Conference_Home'] = data.apply(lambda row: 1 if row.Home in Conferences['Eastern Conference'] else 0, axis=1)
data['Conference'] = data.apply(lambda row: row.Conference_Home-row.Conference_Visitor, axis=1)

data
```

	Date	Visitor	PTSV	Home	PTSH	Overtime	Attend.	Back to back	Div_Visitor	Div_Home	Division	Conference_Visitor	Conference_Home	Conference
0	2019-10-22	New Orleans Pelicans	122	Toronto Raptors	130	1	20,787	0.00	6	1	-5	0	1	1
1	2019-10-22	Los Angeles Lakers	102	Los Angeles Clippers	112	0	19,068	0.00	5	5	0	0	0	0
2	2019-10-23	Chicago Bulls	125	Charlotte Hornets	126	0	15,424	0.00	2	3	1	1	1	0
3	2019-10-23	Detroit Pistons	119	Indiana Pacers	110	0	17,923	0.00	2	2	0	1	1	0
4	2019-10-23	Cleveland Cavaliers	85	Orlando Magic	94	0	18,846	0.00	2	3	1	1	1	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
966	2020-03-10	Brooklyn Nets	104	Los Angeles Lakers	102	0	18,997	0.00	1	5	4	1	0	-1
967	2020-03-11	Detroit Pistons	106	Philadelphia 76ers	124	0	20,172	1.00	2	1	-1	1	1	0
968	2020-03-11	New York Knicks	136	Atlanta Hawks	131	1	15,393	1.00	1	3	2	1	1	0
969	2020-03-11	Charlotte Hornets	109	Miami Heat	98	0	19,6	1.00	3	3	0	1	1	0
970	2020-03-11	Denver Nuggets	97	Dallas Mavericks	113	0	20,302	-1.00	4	6	2	0	0	0

971 rows × 14 columns

Percentatge de victòries en els últims N partits (%W)

La següent funció ens permetrà extreure el % de victòries en els últims N partits de l'equip local i l'equip visitant just abans de començar el partit en questió.

A diferència de les dues altres dades que utilitzen el percentatge de victòries aquesta funció només calcula el percentatge en els últims N partits. Per exemple en el cas que N=10 només tindrem en compte els resultats dels últims 10 partits de cada equip.

El motiu de buscar aquest paràmetre és que permet tenir en compte el moment de forma de cada equip en els últims partits. Ja que si un equip per exemple guanya molts partits al principi de temporada i llavors en perd molts el percentatge de victòries pot resultar enganyós per l'estat de forma actual de l'equip.

Per tal de obtenir aquest percentatge de victòries en els últims N partits haurem de crear els següent diccionari anomenat last_N_games. També crearem dos arrays buits on emmagtzamarem els percentatges de victòries actualitzats i els afegirem al dataframe (last_N_games_visitor, last_N_games_home).

Aquest diccionari contindrà com a clau el nom de l'equip de l'NBA en questió i com a valor associat a la clau una llista amb N strings que seràn 'L' o 'W' en funció de si l'equip ha guanyat o perdut l'últim partit. Un diccionari per N=10 seria del següent format.

{'Los Angeles Lakers': ['W', 'W', 'W', 'L', 'W', 'W', 'W', 'W', 'L']}

No obstant durant els primers 90 partits de la temporada ulitzarem el percentatge de victòries de la temporada anterior que hem guardat al diccionari wins_last_year, d'aquesta manera evitem errors en els primers partits per manca de dades. un cop disputats 90 partits de la temporada cada equip n'haurà disputat com a minim 6 i per tant el percentatge de victories obtingut ja començarà a ser més rellevant (90 partits és un nombre arbitràri que pot ser modificat).

La funció transcorre de la següent manera:

1. Creem un loop amb for i in range( ) de la longitud del dataframe data d'aquesta manera analitzem cada fila que representa un partit.
2. Inicialitzem el diccionari per cada equip si no ha jugat (fins que tots els equips no hagin jugat el seu primer partit el diccionari no estarà complert), també assignem un valor al primer partit de cada equip en aquest cas posarem una llista buida.
3. Apliquem un if per veure si ha guanyat l'equip local o el visitant i dins d'aquest if un altre per veure si estem dins dels primers 90 partits o no.
4. En cas que estiguem dins dels 90 primers partits calcularem el percentatge de victòries a partir de les victòries de la temporada anterior llavors guardarem el resultat del partit com a 'W' o 'L' si l'equip en questió ha guanyat o perdut.
5. En cas que ja s'hagin disputat els primers 90 partits de la temporada, utilitzarem els valors de la llista corresponent a cada equip en què contarem les vegades que hi apareix la string 'W' dividit per la llargada de la llista. Amb això obtindrem el percentatge.
6. Finalment un cop la llista de resultats de cada equip arrivi a la llargada N establerta per la funció borrarem el primer element de la llista. D'aquesta manera sempre tindrem guardats els resultats dels últims N partits.
7. la funció retornarà el diccionari amb el percentatge de victòries de cada equip en els últims N partits i dos arrays que contenen els percentatges de victòries en els últims N partits dels equips locals i visitants.

Finalment restem els dos arrays per tal de obtenir un tercer array que ens doni la diferencia entre els dos percentatges de victories els últims N partits.

```
def last_N_game(data,dim,N):
    last_N_games={}
    last_N_visitor=[]
    last_N_local=[]
    for i in range (dim):
        if data['Visitor'][i] not in last_N_games:
            last_N_games[data['Visitor'][i]]=[]

        if data['Home'][i] not in last_N_games:
            last_N_games[data['Home'][i]]=[]

        if int(data['PTSV'][i])>int(data['PTSH'][i]):
```

```
            if i<90:

                last_N_visitor=np.append(last_N_visitor, "{0:.2f}".format(float(wins_last_year[data['Visitor'][i]])/float(82)))
                last_N_local=np.append(last_N_local, "{0:.2f}".format(float(wins_last_year[data['Home'][i]])/float(82)))
            if i>=90:
                last_N_visitor=np.append(last_N_visitor, "{0:.2f}".format(last_N_games[data['Visitor'][i]].count('W')/len(last_N_games[data['Visitor'][i]])))
                last_N_local=np.append(last_N_local, "{0:.2f}".format(last_N_games[data['Home'][i]].count('W')/len(last_N_games[data['Home'][i]])))
            last_N_games[data['Visitor'][i]]= last_N_games[data['Visitor'][i]]+['W']
            last_N_games[data['Home'][i]]=last_N_games[data['Home'][i]]+['L']
            if len(last_N_games[data['Visitor'][i]])>=N:
                last_N_games[data['Visitor'][i]].pop(0)
                last_N_games[data['Visitor'][i]]=last_N_games[data['Visitor'][i]]
            if len(last_N_games[data['Home'][i]])>=N:
                last_N_games[data['Home'][i]].pop(0)
                last_N_games[data['Home'][i]]=last_N_games[data['Home'][i]]

        else:
            if i<90:

                last_N_visitor=np.append(last_N_visitor, "{0:.2f}".format(float(wins_last_year[data['Visitor'][i]])/float(82)))
                last_N_local=np.append(last_N_local, "{0:.2f}".format(float(wins_last_year[data['Home'][i]])/float(82)))
            if i>=90:
                last_N_visitor=np.append(last_N_visitor, "{0:.2f}".format(last_N_games[data['Visitor'][i]].count('W')/len(last_N_games[data['Visitor'][i]])))
                last_N_local=np.append(last_N_local,"{0:.2f}".format( last_N_games[data['Home'][i]].count('W')/len(last_N_games[data['Home'][i]])))


            last_N_games[data['Visitor'][i]]= last_N_games[data['Visitor'][i]]+['L']
            last_N_games[data['Home'][i]]=last_N_games[data['Home'][i]]+['W']

            if len(last_N_games[data['Visitor'][i]])>=N:
                last_N_games[data['Visitor'][i]].pop(0)
                last_N_games[data['Visitor'][i]]=last_N_games[data['Visitor'][i]]
            if len(last_N_games[data['Home'][i]])>=N:
                last_N_games[data['Home'][i]].pop(0)
                last_N_games[data['Home'][i]]=last_N_games[data['Home'][i]]


        return last_N_games,last_N_visitor,last_N_local

    last_N_g,last_N_v,last_N_l=last_N_game(data,len(data),10)

    last_N=[]
    for i in range(len(last_N_v)):
        last_N=np.append(last_N,"{0:.2f}".format(float(last_N_l[i])-float(last_N_v[i])))

    print(last_N_g)

    data['Last N']=last_N
    data
```

s Pelicans': ['W', 'W', 'L', 'W', 'L', 'L', 'L', 'W', 'W'], 'Toronto Raptors': ['W', 'W', 'L', 'L', 'L', 'W', 'W', 'W', 'W'], 'Los Angeles Lakers': ['W', 'W', 'W', 'L', 'W', 'W', 'W', 'W',

ate	Visitor	PTSV	Home	PTSH	Overtime	Attend.	Back to back	Div_Visitor	Div_Home	Division	Conference_Visitor	Conference_Home	Conference	Last N
10-22	New Orleans Pelicans	122	Toronto Raptors	130	1	20,787	0.00	6	1	-5	0	1	1	0.31
10-22	Los Angeles Lakers	102	Los Angeles Clippers	112	0	19,068	0.00	5	5	0	0	0	0	0.14
10-23	Chicago Bulls	125	Charlotte Hornets	126	0	15,424	0.00	2	3	1	1	1	0	0.21
10-23	Detroit Pistons	119	Indiana Pacers	110	0	17,923	0.00	2	2	0	1	1	0	0.09
10-23	Cleveland Cavaliers	85	Orlando Magic	94	0	18,846	0.00	2	3	1	1	1	0	0.28
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
03-10	Brooklyn Nets	104	Los Angeles Lakers	102	0	18,997	0.00	1	5	4	1	0	-1	0.45
-11	Detroit Pistons	106	Philadelphia 76ers	124	0	20,172	1.00	2	1	-1	1	1	0	0.33
-11	New York Knicks	136	Atlanta Hawks	131	1	15,393	1.00	1	3	2	1	1	0	0.11
-11	Charlotte Hornets	109	Miami Heat	98	0	19,6	1.00	3	3	0	1	1	0	0.34
-11	Denver Nuggets	97	Dallas Mavericks	113	0	20,302	-1.00	4	6	2	0	0	0	0.00

columns

Càlcul del Net Rating(diferència de punts anotats a favor i en contra)

La següent funció ens permetrà extreure el paràmetre anomanet Net Rating, aquest paràmetre és el valor que té en compte la diferència entre punts a favor i en contra de cada equip dividit per el nombre de partits totals disputats. Això permet veure si un equip sol guanyar o perdre per molts o pocs punts de diferència.

Per tal de obtenir aquesta diferència mitjana de punts per partit crearem el següent diccionari anomenat net_rat={}. També crearem dos arrays buits on emmagtzamarem el net rating tant per l'equip local com per l'equip visitant (net_rat_visitor, net_rat_local).

El diccionari tindrà com a clau el nom de l'equip de l'NBA en questió i com a valor associat a la clau una llista de dos valors el primer contindrà la resta de punts a favor menys punts en contra, el segon valor serà el nombre de partits disputats, a continució en veiem un exemple:

{New York Knicks': [-417, 66]}

La funció transcorre de la següent manera:

1. Creem un loop amb for i in range( ) de la longitud del dataframe data d'aquesta manera analitzem cada fila que representa un partit de la temporada.
2. Inicialitzem el diccionari per cada equip si no ha jugat utilitzant una llista del següent format [0,1] (fins que tots els equips no hagin jugat el seu primer partit el diccionari no estarà complert).
3. Un cop inicilitzats tots els diccionaris creem una variable, anomenada dif que conté la diferència de punts del partit analitzat, tot seguit calcularem el net rating i el guardarem a les arrays.

4. Finalment actualitzarem els valors de la llista del diccionari per cada equip. Tant de la diferència de punts com la de partits disputats.
5. La funció retorna el valor de les dues arrays i també el diccionari que conté els valors necessaris per calcular el net rating.

Finalment restem les dues arrays per tal de obtenir un tercer array que ens doni la diferencia entre el net rating dels dos equips.

```
def net_rating(data,dim):
    net_rat={}
    net_rat_visitor=[]
    net_rat_local=[]
    for i in range(dim):
        if data['Visitor'][i] not in net_rat:
            net_rat[data['Visitor'][i]]=[0,1]
            net_rat_visitor=np.append(net_rat_visitor,"{0:.2f}".format(0.0))
        if data['Home'][i] not in net_rat:
            net_rat[data['Home'][i]]=[0,1]
            net_rat_local=np.append(net_rat_local, "{0:.2f}".format(0.0))
        else:
            dif=int(data['PTSV'][i])-int(data['PTSH'][i])

            net_rat_visitor=np.append(net_rat_visitor,"{0:.2f}".format(int(net_rat[data['Visitor'][i]][0])/int(net_rat[data['Visitor'][i]][1])))
            net_rat_local=np.append(net_rat_local,"{0:.2f}".format(int(net_rat[data['Home'][i]][0])/int(net_rat[data['Home'][i]][1])))
            if dif<0:
                dif=dif*-1
            if int(data['PTSV'][i])>int(data['PTSH'][i]):
                net_rat[data['Visitor'][i]][0]=net_rat[data['Visitor'][i]][0]+dif
                net_rat[data['Visitor'][i]][1]=net_rat[data['Visitor'][i]][1]+1
                net_rat[data['Home'][i]][0]=net_rat[data['Home'][i]][0]+(dif*-1)
                net_rat[data['Home'][i]][1]=net_rat[data['Home'][i]][1]+1
            if int(data['PTSV'][i])<int(data['PTSH'][i]):
                net_rat[data['Visitor'][i]][0]=net_rat[data['Visitor'][i]][0]+(dif*-1)
                net_rat[data['Visitor'][i]][1]=net_rat[data['Visitor'][i]][1]+1
                net_rat[data['Home'][i]][0]=net_rat[data['Home'][i]][0]+dif
                net_rat[data['Home'][i]][1]=net_rat[data['Home'][i]][1]+1

    return net_rat,net_rat_visitor,net_rat_local

net_rating,net_rating_visitor,net_rating_local=net_rating(data,len(data))

print(net_rating)

net_rating=[]
for i in range(len(net_rating_local)):
    net_rating=np.append(net_rating,"{0:.3f}".format(float(net_rating_local[i])-float(net_rating_visitor[i])))


data['Net Rating']=net_rating
data
```

ıs Pelicans': [-45, 64], 'Toronto Raptors': [405, 64], 'Los Angeles Lakers': [477, 63], 'Los Angeles Clippers': [388, 63], 'Chicago Bulls': [-199, 65], 'Charlotte Hornets': [-440, 65], 'Det

ıte	Visitor	PTSV	Home	PTSH	Overtime	Attend.	Back to back	Div_Visitor	Div_Home	Division	Conference_Visitor	Conference_Home	Conference	Last N	Net Rating
10-22	New Orleans Pelicans	122	Toronto Raptors	130	1	20,787	0.00	6	1	-5	0	1	1	0.31	0.000
10-22	Los Angeles Lakers	102	Los Angeles Clippers	112	0	19,068	0.00	5	5	0	0	0	0	0.14	0.000
10-23	Chicago Bulls	125	Charlotte Hornets	126	0	15,424	0.00	2	3	1	1	1	0	0.21	0.000
10-23	Detroit Pistons	119	Indiana Pacers	110	0	17,923	0.00	2	2	0	1	1	0	0.09	0.000
10-23	Cleveland Cavaliers	85	Orlando Magic	94	0	18,846	0.00	2	3	1	1	1	0	0.28	0.000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
03-10	Brooklyn Nets	104	Los Angeles Lakers	102	0	18,997	0.00	1	5	4	1	0	-1	0.45	8.400
03-11	Detroit Pistons	106	Philadelphia 76ers	124	0	20,172	1.00	2	1	-1	1	1	0	0.33	5.230
03-11	New York Knicks	136	Atlanta Hawks	131	1	15,393	1.00	1	3	2	1	1	0	0.11	-1.410
03-	Charlotte Hornets	109	Miami Heat	98	0	19,6	1.00	3	3	0	1	1	0	0.34	10.210

Càlcul de si s'ha jugat pròrroga a l'últim partit disputat

La següent funció ens permetrà extreure un paràmetre que inidicarà si l'equip en questió ha disputat una pròrroga a l'últim partit o no.

Primer de tot cal inidicar que els partits a la NBA duren 48 minuts i que en cas d'empat les pròrrogues són de 5 minuts extres. En cas de pròrroga això suposa un desgast extra per l'equip en questió que pot afectar al seu rendiment al partit següent.

Per poder obtenir aquest paràmetre s'ha creat un diccionari anomenat overtime que conté com a clau el nom de l'equip i com a valor associat un 0 si no ha disputat pròrroga l'últim parti i un 1 si l'ha disputat.

{'Toronto Raptors': 0, 'New Orleans Pelicans': 0, 'Los Angeles Clippers': 0, 'Los Angeles Lakers': 0, 'Charlotte Hornets': 0, 'Chicago Bulls': 0, 'Indiana Pacers': 0, 'Detroit Pistons': 0, 'Orlando Magic': 0, 'Cleveland Cavaliers': 0, 'Brooklyn Nets': 0, 'Minnesota Timberwolves': 0, 'Miami Heat': 0, 'Memphis Grizzlies': 0, 'Philadelphia 76ers': 0, 'Boston Celtics': 0, 'Dallas Mavericks': 0, 'Washington Wizards': 0, 'San Antonio Spurs': 0, 'New York Knicks': 1, 'Utah Jazz': 0, 'Oklahoma City Thunder': 0, 'Phoenix Suns': 0, 'Sacramento Kings': 0, 'Portland Trail Blazers': 0, 'Denver Nuggets': 0, 'Atlanta Hawks': 1, 'Houston Rockets': 0, 'Milwaukee Bucks': 0, 'Golden State Warriors': 0}

També crearem dues array overtime_visitor i overtime_local on guardarem els valors.

La funció transcorre de la següent manera:

1. Creem un loop amb for i in range( ) de la longitud del dataframe data d'aquesta manera analitzem cada fila que representa un partit de la temporada.
2. Per cada equip guardem el valor associat de al diccionari i posteriorment modificarem el valor associat en funció de si l'equip ha disputat pròrroga en el partit actual.

3. Finalment la funció retorna el diccionari overtime i les arrays overtime_local i overtime_visitor.

Per acabar restem les dues arrays per tal d'obtenir un tercer array que ens doni la diferència entre els dos equips pel que fa a pròrroga disputada. Si el resultat de la resta és 0 voldrà dir que ambdós equips han jugat pròrroga el partit anterior o cap dels dos l'ha disputat. Si la resta dóna com a resultat +1 o -1 voldrà dir que són un equip ha disputat pròrroga i l'altre no.

```
def OT_last_match(data,dim):
    overtime={}
    overtime_local=[]
    overtime_visitor=[]
    for i in range (dim):
        if  data['Home'][i] not in overtime:
            overtime[data['Home'][i]]=0
        if  data['Visitor'][i] not in overtime:
            overtime[data['Visitor'][i]]=0
        overtime_local=np.append(overtime_local,overtime[data['Home'][i]])
        overtime_visitor=np.append(overtime_visitor,overtime[data['Visitor'][i]])
        if data['Overtime'][i]==1:
            if overtime[data['Home'][i]]==0:
                overtime[data['Home'][i]]=overtime[data['Home'][i]]+1
            if overtime[data['Visitor'][i]]==0:
                overtime[data['Visitor'][i]]=overtime[data['Visitor'][i]]+1

        if data['Overtime'][i]==0 and overtime[data['Home'][i]]==1:
            overtime[data['Home'][i]]=overtime[data['Home'][i]]-1
        if data['Overtime'][i]==0 and overtime[data['Visitor'][i]]==1:
            overtime[data['Visitor'][i]]=overtime[data['Visitor'][i]]-1
    return(overtime,overtime_local,overtime_visitor)

OTs,OT_local,OT_visit=OT_last_match(data,len(data))
#print(OTs,OT_local,OT_visit)

OT=[]
for i in range(len(OT_local)):
    OT=np.append(OT,"{0:.2f}".format(float(OT_local[i])-float(OT_visit[i])))
print(OTs)
data['OT last match']=OT
data
```

```
: 0, 'New Orleans Pelicans': 0, 'Los Angeles Clippers': 0, 'Los Angeles Lakers': 0, 'Charlotte Hornets': 0, 'Chicago Bulls': 0, 'Indiana Pacers': 0, 'Detroit Pistons': 0, 'Orlando Magic':
```

Visitor	PTSV	Home	PTSH	Overtime	Attend.	Back to back	Div_Visitor	Div_Home	Division	Conference_Visitor	Conference_Home	Conference	Last N	Net Rating	OT last match
New Orleans Pelicans	122	Toronto Raptors	130	1	20,787	0.00	6	1	-5	0	1	1	0.31	0.000	0.00
Los Angeles Lakers	102	Los Angeles Clippers	112	0	19,068	0.00	5	5	0	0	0	0	0.14	0.000	0.00
Chicago Bulls	125	Charlotte Hornets	126	0	15,424	0.00	2	3	1	1	1	0	0.21	0.000	0.00
etroit Pistons	119	Indiana Pacers	110	0	17,923	0.00	2	2	0	1	1	0	0.09	0.000	0.00
Cleveland Cavaliers	85	Orlando Magic	94	0	18,846	0.00	2	3	1	1	1	0	0.28	0.000	0.00
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

Jugadors que són estrelles de la NBA

A la NBA hi ha jugadors que tenen influència important en els partits i que la seva simple presència en el partit o no pot canviar les possibilitats d'un equip de guanyar un partit. Aquests jugadors s'han escollit a partir de una llista els 50 millors jugadors de la lliga que es fa al principi de cada temporada.

Aquesta funció seleccionarà entre el dataframe que conté tots els jugadors que han disputat de la temporada els que tenen un 1 en la columna STAR i els emmagatzema en un diccionari.

Crearem un diccionari anomenat STARS que contindrà com a clau el nom dels jugadors que són considerats entre els 50 millors a l'inici de la temporada i com a valor associat un 1.

La funció transcorre de la següent manera:

1. Creem un loop amb for i in range( ) de la longitud del dataframe data d'aquesta manera analitzem cada fila que representa un jugador de la NBA.
2. Guardarem tal com hem explicat els jugadors que són considerats entre els 50 millors.
3. Finalment la funció retorna el diccionari STARS.

```
team_capitals={'Toronto Raptors':'TOR', 'Boston Celtics':'BOS', 'Philadelphia 76ers':'PHI', 'Brooklyn Nets':'BRK', 'New York Knicks':'NYK','Milwaukee Bucks':'MIL', 'Indiana Pacers'

def player_stat(data,dim):
    stars={}
    for i in range (dim):
        if data['STAR'][i]==1:
            stars[data['Player'][i]]=data['STAR'][i]
    return stars


stars=player_stat(data_players,len(data_players))
print(stars)
```

```
{'James Harden': 1, 'Giannis Antetokounmpo': 1, 'LeBron James': 1, 'Nikola Jokic': 1, 'Anthony Davis': 1, 'Damian Lillard': 1, 'Luka Doncic': 1, 'Kawhi Leonard': 1, 'Jimmy Butler': 1, 'Chr
```

Partits disputats per els jugadors que són estrelles de la NBA

A la NBA hi ha jugadors que tenen influència important en els partits i que la seva simple presència en el partit o no pot canviar les possibilitats d'un equip de guanyar un partit. Per tal de saber quins partits han jugat i quins han sigut baixa realitzarem la següent funció.

Aquesta funció rep com a paràmetre el diccionari creat a la funció anterior. El que farem serà crear un diccionari que contingui com a clau el nom del jugador i com a valor associat una llista amb la data de tots els partits disputats per cada jugador durant la temporada. Aquesta dada la treurem de les estadístiques individuals de cada jugador per cada partit.

Crearem un diccionari anomenat games_players on guardarem les dades mencionades anteriorment.

La funció transcorre de la següent manera:

1. Creem un loop amb for key in dic d'aquesta manera buscarem els partits disputats per cada jugador ja que cada key representa un jugador de la NBA considerat estrella.
2. Es crea un dataframe dins del primer loop amb les estadístiques de cada partit del jugador analitzat.
3. Tot seguit aplicarem un loop amb for i in range amb la len d'aquest dataframe i n'extreurem una llista amb totes les dates dels partits que ha disputat i l'afegirem al diccionari games_players amb el jugador corresponent
4. Finalment la funció retorna el diccionari games_players, es pot veure el diccionari que s'en obté a continuació.

```
def dic_games_played(dic):
    games_players={}
    for key in dic:
        games=[]
        players=pd.read_csv(io.BytesIO(uploaded["games_"+key+".csv"]))

        for i in range (len(players)):
            match=players['GAME_DATE'][i]
            date=pd.to_datetime(match, format='%b %d, %Y')
            games=games+[date]
        games_players[key]=games
    return games_players

dic_games_played=dic_games_played(stars)
print(dic_games_played)
```

```
{'James Harden': [Timestamp('2020-03-10 00:00:00'), Timestamp('2020-03-08 00:00:00'), Timestamp('2020-03-07 00:00:00'), Timestamp('2020-03-05 00:00:00'), Timestamp('2020-03-02 00:00:00'),
```

Diccionari que conté els jugadors estrella i el seu equip (incloent traspassos)

Ens cal assignar també cada jugador al equip al que pertany. Amb aquest objectiu volem obtenir un diccionari que tingui com a clau el nom del jugador i com a valor el nom de l'equip on juga.

No obstant a la NBA es poden realitzar fitxatges durant la temporada el que suposa un canvi d'equip. En cas que això es produeixi ho emmagtzamarem amb una llista associada a la clau que contindrà l'equip que pertanyia abans del fitxatge, l'equip pel que ha fitxat i la data en

la que s'ha produit el traspàs.

{"D'Angelo Russell": ['MIN', 'GSW', 'FEB 05, 2020']}

Per obtenir aquest diccionari ho farem de la següent manera:

Crearem un diccionari anomenat players_teams on guardarem les dades mencionades anteriorment.

La funció transcorre de la següent manera:

1. Creem un loop amb for i in range del dataframe data_players que conté les estadístiques globals de tots els jugadors de la lliga al llarg de la temporada.
2. Si l'abreviatura de l'equip al que pertany al jugador és 'TOT' que significa to other team, consultem les seves estadístiques de cada partit per veure quan es va produir el canvi d'equip i quan veiem el dia aplicant un while en guardem la data a la llista de la forma que hem explicat anteriorment.
3. En cas que el jugador hagi jugat tota la temporada al mateix equip, guardem simplement la abreviatura del nom de l'equip en el que ha jugat.
4. Finalment la funció retorna el diccionari players_teams, es pot veure el diccionari que s'en obté a continuació.

```
def player_team(data,dim,dic):
    players_teams={}
    for i in range(dim):
        last_team=[]
        if data['Player'][i] in stars and data['Tm'][i] == 'TOT':

            j=0
            players= pd.read_csv(io.BytesIO(uploaded["games_"+data['Player'][i]+".csv"]))
            while j+1 < (len(players)):
                j=j+1
                if players['MATCHUP'][j][:3]!=players['MATCHUP'][j-1][:3]:
                    date=players['GAME_DATE'][j]
                    date=pd.to_datetime(date, format='%b %d, %Y')
                    last_team=last_team+[players['MATCHUP'][j-1][:3],players['MATCHUP'][j][:3],date]

            players_teams[data['Player'][i]]=last_team


        if data['Player'][i] in stars and data['Tm'][i] not in players_teams and data['Tm'][i] != 'TOT' :
            players_teams[data['Player'][i]]=[data['Tm'][i]]

    return players_teams


play_team=player_team(data_players,len(data_players),dic_games_played)
print(play_team)
```

```
{'James Harden': ['HOU'], 'Giannis Antetokounmpo': ['MIL'], 'LeBron James': ['LAL'], 'Nikola Jokic': ['DEN'], 'Anthony Davis': ['LAL'], 'Damian Lillard': ['POR'], 'Luka Doncic': ['DAL'], '
```

Paràmetre d'absència de les estrelles de l'equip

La següent funció ens permetrà extreure el paràmetre d'absència de les estrelles dels dos equips que disputen el partit. Totes les funcions anteriors estan orientades a poder obtenir aquest paràmetre. Bàsicament indicarà si algun dels jugadors que pertanyen al top 50 no ha disputat el partit en questió. Això ens servirà per analitzar amb detall l'efecte de les absències dels jugadors considerats estrelles en cada partit.

Per tal d'obtenir aquest paràmetre crearem dos arrays buits on emmagtzamarem les baixes de les estrelles tant per l'equip local com per l'equip visitant (stars_inj_visitor, stars_inj_local).

La funció transcorre de la següent manera:

1. Creem un loop amb for i in range( ) de la longitud del dataframe data d'aquesta manera analitzem cada fila que representa un partit de la temporada.
2. Per cada jugador del diccionari que conté el Top 50 que hem creat anteriorment. Analitzarem si aquest jugador ha estat trespassat o no durant la temporada.
3. En cas que hagi estat trespassat consultarem si la data del partit analitzat és anterior o posterior a la del seu traspàs. D'aquesta manera poder saber en quin equip jugava en la data d'aquell partit. A continuació es mira si el jugador considerat top 50 ha jugat el partit en questió o no i en cas que així sigui guardarem el paràmetre que definirà la seva absència que serà un 1.
4. En cas que no hagi estat traspassat mirarem directament si el jugador considerat top 50 ha jugat el partit en questió o no i en cas que així sigui guardarem el paràmetre que definirà la seva absència que serà un 1.
5. A continuació guardem els valors acumulats per cada equip a les arrays creades inicialment i la funció les retorna.

Finalment restem les dues arrays per tal de obtenir un tercer array que ens doni la diferencia entre els paràmetres d'absència dels dos equips.

```
def stars_rest_inj(data_players,dim_players,players,data,dim,stat):
    stars_inj_local=[]
    stars_inj_visitor=[]

    for i in range (dim):

        value_local=0
        value_visitor=0
        for key in players:

            if len(play_team[key])==3:

                if data['Date'][i]<play_team[key][2] and data['Date'][i] not in players[key]and play_team[key][1] == team_capitals[data['Home'][i]]:

                    value_local=value_local+stat[key]
                if data['Date'][i]<play_team[key][2]==False and data['Date'][i] not in players[key] and play_team[key][0] == team_capitals[data['Home'][i]]:

                    value_local=value_local+stat[key]

                if data['Date'][i]<play_team[key][2] and data['Date'][i] not in players[key]and play_team[key][1] == team_capitals[data['Visitor'][i]]:

                    value_visitor=value_local+stat[key]
                if data['Date'][i]<play_team[key][2]==False and data['Date'][i] not in players[key] and play_team[key][0] == team_capitals[data['Visitor'][i]]:

                    value_visitor=value_local+stat[key]
```

```
            else:

                if data['Date'][i] not in players[key] and play_team[key][0] == team_capitals[data['Home'][i]]:

                    value_local=value_local+stat[key]

                if data['Date'][i] not in players[key] and play_team[key][0] == team_capitals[data['Visitor'][i]]:
                    value_visitor=value_visitor+stat[key]

        stars_inj_local=np.append(stars_inj_local,"{0:.2f}".format(value_local))
        stars_inj_visitor=np.append(stars_inj_visitor,"{0:.2f}".format(value_visitor))


    return stars_inj_local,stars_inj_visitor

stars_inj_loc,stars_inj_visit=stars_rest_inj(data_players,len(data_players),dic_games_played,data,len(data),stars)
stars_inj=[]
for j in range(len(stars_inj_loc)):
    stars_inj=np.append(stars_inj,"{0:.2f}".format(float(stars_inj_loc[j])-float(stars_inj_visit[j])))
data['Rest/Injuries Visitor Stars']=stars_inj_visit
data['Rest/Injuries Home Stars']=stars_inj_loc
data['Rest/Injuries Stars']=stars_inj
data
#print(stars_inj_loc)
#print(stars_inj_visit)
```

	ate	Visitor	PTSV	Home	PTSH	Overtime	Attend.	Back to back	Div_Visitor	Div_Home	Division	Conference_Visitor	Conference_Home	Conference	Last N	Net Rating	OT last match	Rest/Injuries Visitor Stars	Rest/Inju Home S
	019-	New Orleans	122	Toronto	130	1	20.787	0.00	6	1	-5	0	1	1	0.31	0.000	0.00	0.00	

Ratxa de N victòries o derrotes

La següent funció ens permetrà extreure la ratxa de victòries en els últims n partits de l'equip local i l'equip visitant just abans de començar el partit en questió.

Aquesta funció calcula la ratxa de partits guanyats o perduts consecutius. Per exemple en el cas que n=10 només tindrem en compte només les ratxes de 10 o més victòries o derrotes. És a dir si un equip ha guanyat 11 partits seguits l'equip rebrà un paràmetre 11 si un equip ha guanyat 4 seguits el valor serà 0 ja que està per sota del llindar fixat per N i si un equip ha perdut 12 consecutius el paràmetres serà -12 amb signe negatiu per indicar que es tracta d'una derrota.

El motiu de buscar aquest paràmetre és que permet tenir en compte el moment de forma de cada equip ja que cal tenir en compte si un equip porta una mala o bona dinàmica en els últims partits. Aquest fet pot tenir una influència important en el resultat i cal tenir-la en compte. Una altre cosa que s'ha de tenir en compte és a partir de quants N partits es considera una ratxa.

Per tal de obtenir aquesta ratxa de victòries haurem de crear els següent diccionari anomenat streak_N_games. També crearem dos arrays buits on emmagtzamarem els valors de les ratxes i els afegirem al dataframe (streak_N_visitor, streak_N_home).

Aquest diccionari contindrà com a clau el nom de l'equip de l'NBA en questió i com a valor associat a la clau una llista amb N strings que seràn 'L' o 'W' en funció de si l'equip ha guanyat o perdut. Un diccionari per N=30 seria del següent format.

{'New Orleans Pelicans': ['W', 'L', 'W', 'W', 'L', 'W', 'W', 'L', 'W', 'L', 'L', 'W', 'W', 'W', 'L', 'L', 'W', 'W', 'W', 'L', 'W', 'W', 'L', 'W', 'L', 'L', 'L', 'W', 'W']

La funció transcorre de la següent manera:

1. Creem un loop amb for i in range( ) de la longitud del dataframe data d'aquesta manera analitzem cada fila que representa un partit.
2. Inicialitzem el diccionari per cada equip si encara no ha jugat (fins que tots els equips no hagin jugat el seu primer partit el diccionari no estarà complert), també assignem un valor al primer partit de cada equip en aquest cas posarem un 0 ja que necessitem que la llista contingui algun valor.
3. Analitzem el diccionari per local i visitant i per ratxa de victòries i de derrotes. En cada un dels casos apliquem un loop a la llista per contar del últims partits quina és la ratxa amb un contador. En cas que la ratxa superi el llindar n, afegim a les arrays el valor d'aquesta ratxa, en cas contrari i afegirem un 0 .
4. Un cop contat el valor de la ratxa actualitzem el valor del diccionari amb una 'W' o una 'L' en funció del resultat del partit.
5. La funció retornarà el diccionari amb la ratxa de victòries o derrotes de cada equip més grans o iguals que n i dos arrays que contenen les ratxes de victòries i derrotes dels equips locals i visitants.

Finalment restem els dos arrays per tal de obtenir un tercer array que ens doni la diferencia entre les ratxes de victories i derrotes d'ambdós equips.

```
def streak(data,dim,N,n):
    streak_N_games={}
    streak_N_visitor=[]
    streak_N_local=[]
```

```
        for i in range (dim):

            if data['Visitor'][i] not in streak_N_games:
               streak_N_games[data['Visitor'][i]]=[0]
               streak_N_visitor=np.append(streak_N_visitor, 0)

            if data['Home'][i] not in streak_N_games:
                streak_N_games[data['Home'][i]]=[0]
                streak_N_local=np.append(streak_N_local, 0)
            wins=0
            lost=0
            if streak_N_games[data['Visitor'][i]][-1]=='W':

                for j in range(len(streak_N_games[data['Visitor'][i]])):
                    if streak_N_games[data['Visitor'][i]][-j-1]=='W':
                        wins=wins+1
                    else:
                        break
                if wins>=n:
                    streak_N_visitor=np.append(streak_N_visitor, wins)
                else:
                    streak_N_visitor=np.append(streak_N_visitor, 0)

            if streak_N_games[data['Visitor'][i]][-1]=='L':

                for j in range(len(streak_N_games[data['Visitor'][i]])):
                    if streak_N_games[data['Visitor'][i]][-j-1]=='L':
                        lost=lost+1
                    else:
                        break
                if lost>=n:
                    streak_N_visitor=np.append(streak_N_visitor, -lost)
                else:
                    streak_N_visitor=np.append(streak_N_visitor, 0)
        wins=0
        lost=0
        if streak_N_games[data['Home'][i]][-1]=='W':

            for j in range(len(streak_N_games[data['Home'][i]])):
                if streak_N_games[data['Home'][i]][-j-1]=='W':
                    wins=wins+1
                else:
                    break
            if wins>=n:
                streak_N_local=np.append(streak_N_local, wins)
            else:
                streak_N_local=np.append(streak_N_local, 0)

        if streak_N_games[data['Home'][i]][-1]=='L':

            for j in range(len(streak_N_games[data['Home'][i]])):
                if streak_N_games[data['Home'][i]][-j-1]=='L':
                    lost=lost+1
                else:
                    break
```

```
            if lost>=n:
                streak_N_local=np.append(streak_N_local, -lost)
            else:
                streak_N_local=np.append(streak_N_local, 0)


        if int(data['PTSV'][i])>int(data['PTSH'][i]):
            streak_N_games[data['Visitor'][i]]= streak_N_games[data['Visitor'][i]]+['W']
            streak_N_games[data['Home'][i]]=streak_N_games[data['Home'][i]]+['L']
            if len(streak_N_games[data['Visitor'][i]])>=N:
                streak_N_games[data['Visitor'][i]].pop(0)
                streak_N_games[data['Visitor'][i]]=streak_N_games[data['Visitor'][i]]
            if len(streak_N_games[data['Home'][i]])>=N:
                streak_N_games[data['Home'][i]].pop(0)
                streak_N_games[data['Home'][i]]=streak_N_games[data['Home'][i]]

        else:
            streak_N_games[data['Visitor'][i]]= streak_N_games[data['Visitor'][i]]+['L']
            streak_N_games[data['Home'][i]]=streak_N_games[data['Home'][i]]+['W']

            if len(streak_N_games[data['Visitor'][i]])>=N:
                streak_N_games[data['Visitor'][i]].pop(0)
                streak_N_games[data['Visitor'][i]]=streak_N_games[data['Visitor'][i]]
            if len(streak_N_games[data['Home'][i]])>=N:
                streak_N_games[data['Home'][i]].pop(0)
                streak_N_games[data['Home'][i]]=streak_N_games[data['Home'][i]]

        return streak_N_games,streak_N_visitor,streak_N_local

    streak_N_g,streak_N_v,streak_N_l=streak(data,len(data),30,8)

    print(streak_N_g)
    streak_N=[]
    for i in range(len(streak_N_l)):
        streak_N=np.append(streak_N,"{0:.2f}".format(float(streak_N_l[i])-float(streak_N_v[i])))

    data['Win Streak']=streak_N
    data
```

L', 'W', 'W', 'W', 'L', 'L', 'W', 'W', 'W', 'L', 'W', 'W', 'L', 'W', 'L', 'L', 'L', 'W', 'W'], 'Toronto Raptors': ['W', 'L', 'W', 'L', 'W', 'W', 'W', 'W', 'W', 'W', 'W', 'W', 'W', 'W', 'W'

sitor	Div_Home	Division	Conference_Visitor	Conference_Home	Conference	Last N	Net Rating	OT last match	Rest/Injuries Visitor Stars	Rest/Injuries Home Stars	Rest/Injuries Stars	Win Streak
6	1	-5	0	1	1	0.31	0.000	0.00	0.00	0.00	0.00	0.00
5	5	0	0	0	0	0.14	0.000	0.00	0.00	1.00	1.00	0.00
2	3	1	1	1	0	0.21	0.000	0.00	0.00	0.00	0.00	0.00
2	2	0	1	1	0	0.09	0.000	0.00	1.00	1.00	0.00	0.00
2	3	1	1	1	0	0.28	0.000	0.00	0.00	0.00	0.00	0.00
...	...	...	...	...	...	...	...	...	...	...	...	...

Distància recorreguda per cada equip

Aquesta funció té com a objectiu calcular la distància en Km recorreguda per cada equip anteriorment al partit analitzat. D'aquesta manera som capaços de monitoritzar quin dels dos equips ha recorregut menys Km i per tant arriva menys fatigat al partit. Pot ser un factor rellevant i tenir un efecte similar al de la funció del Back to Back que hem fet anteriorment.

Cal destacar que Estats Units és un país enorme amb grans distàncies entre les ciutats on es disputen els partits per exemple la distància entre Los Àngeles i Boston és de 4800 km. Per tant al disputar tants partits en pocs dies els equips passen una part important del seu temps de descans a l'avió. Per tots aquests motius s'ha realitzat la funció següent.

Crearem 3 diccionaris un es dirà travels i contindrà les distàncies que ja hem obtingut anteriorment al Dataframe de distàncies entre ciutats. La clau serà la ciutat origen i la ciutat destí i el valor associat un float que contindrà la distància en Km. El diccionari last_match guardarà la ciutat on s'hagi disputat l'últim partit per cada equip i el diccionari kilometers guardarà les distàncies acumulades totals per cada equip. També crearem com sempre dos arrays per guardar els km de l'equip local i visitant

La funció transcorre de la següent manera:

1. Creem un loop amb for i in range( ) de la longitud del dataframe data d'aquesta manera analitzem cada fila que representa un partit.
2. Inicialitzem els diccionaris kilometers i last_match amb els primers partits de cada equip i en el cas de l'equip guardarem la distància entre la seva ciutat i la ciutat on han disputat el partit en km. En canvi per els equips que juguen a casa guardarem lògicament un 0.
3. Un cop inicialitzats els diccionaris procedim a buscar per cada partit la distància acumulada per l'equip local i visitant en l'últim desplaçament i l'afegim al diccionari kilometers i last_match. Finalment guardem aquest valor actulitzat a les arrays corresponents. Cal dir que lògicament si un equip juga 2 o més partits a casa la distància nova serà 0 ja que a last match tindrà guardat el nom de la seva ciutat i quan es busqui la distància entre les dues ciutats al ser la mateixa el diccionari travels contindrà un 0.

Finalment restem els dos arrays per tal de obtenir un tercer array que ens doni la diferencia entre les distàncies recorregudes per ambdós equips.

```
def traveling(data,dim,data_distance,dim_distance):
    travels={}
    for i in range(dim_distance):
        travels[str(data_distance['City Origin'][i])+' - '+str(data_distance['City Destiny'][i])]=data_distance['Distance (Km)'][i]

    kilometers={}
    last_match={}
    km_local=[]
    km_visitor=[]
    for i in range(dim):

        if data['Home'][i] not in kilometers:
            kilometers[data['Home'][i]]=0
            km_local=np.append(km_local,"{0:.2f}".format(float(kilometers[data['Home'][i]])))
        if data['Home'][i] not in last_match:
            last_match[data['Home'][i]]=data['Home'][i]
        if data['Visitor'][i] not in kilometers:
            kilometers[data['Visitor'][i]]=float(travels[str(data['Home'][i])+' - '+str(data['Visitor'][i])])
        if data['Visitor'][i] not in last_match:
            last_match[data['Visitor'][i]]=data['Home'][i]
            km_visitor=np.append(km_visitor,"{0:.2f}".format(float(kilometers[data['Visitor'][i]])))


        else:
            if  last_match[data['Home'][i]]==data['Home'][i]:
                kilometers[data['Visitor'][i]]=kilometers[data['Visitor'][i]]+float(travels[str(data['Home'][i])+' - '+str(last_match[data['Visitor'][i]])])
                kilometers[data['Home'][i]]=kilometers[data['Home'][i]]*0.5
                last_match[data['Visitor'][i]]=data['Home'][i]
                last_match[data['Home'][i]]=data['Home'][i]
                km_local=np.append(km_local,"{0:.2f}".format(float(kilometers[data['Home'][i]])))
                km_visitor=np.append(km_visitor,"{0:.2f}".format(float(kilometers[data['Visitor'][i]])))
            else:
                kilometers[data['Visitor'][i]]=kilometers[data['Visitor'][i]]+float(travels[str(data['Home'][i])+' - '+str(last_match[data['Visitor'][i]])])
                kilometers[data['Home'][i]]=kilometers[data['Home'][i]]+float(travels[str(data['Home'][i])+' - '+str(last_match[data['Home'][i]])])
                last_match[data['Visitor'][i]]=data['Home'][i]
                last_match[data['Home'][i]]=data['Home'][i]
                km_local=np.append(km_local,"{0:.2f}".format(float(kilometers[data['Home'][i]])))
                km_visitor=np.append(km_visitor,"{0:.2f}".format(float(kilometers[data['Visitor'][i]])))




    return km_local,km_visitor,last_match,kilometers


km_l,km_visit,last_m,kilo=traveling(data,len(data),data_distance,len(data_distance))

km=[]
for i in range(len(km_l)):
    km=np.append(km,"{0:.2f}".format(float(km_l[i])-float(km_visit[i])))

data['Km']=km
data
```

SV	Home	PTSH	Overtime	Attend.	Back to back	Div_Visitor	Div_Home	Division	Conference_Visitor	Conference_Home	Conference	Last N	Net Rating	OT last match	Rest/Injuries Visitor Stars	Rest/Injuries Home Stars	Rest/Injuri Sta
22	Toronto Raptors	130	1	20,787	0.00	6	1	-5	0	1	1	0.31	0.000	0.00	0.00	0.00	0
02	Los Angeles Clippers	112	0	19,068	0.00	5	5	0	0	0	0	0.14	0.000	0.00	0.00	1.00	1
25	Charlotte Hornets	126	0	15,424	0.00	2	3	1	1	1	0	0.21	0.000	0.00	0.00	0.00	0
19	Indiana Pacers	110	0	17,923	0.00	2	2	0	1	1	0	0.09	0.000	0.00	1.00	1.00	0
35	Orlando Magic	94	0	18,846	0.00	2	3	1	1	1	0	0.28	0.000	0.00	0.00	0.00	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
04	Los Angeles Lakers	102	0	18,997	0.00	1	5	4	1	0	-1	0.45	8.400	0.00	1.00	0.00	-1
06	Philadelphia 76ers	124	0	20,172	1.00	2	1	-1	1	1	0	0.33	5.230	0.00	1.00	1.00	0
36	Atlanta Hawks	131	1	15,393	1.00	1	3	2	1	1	0	0.11	-1.410	1.00	0.00	0.00	0
09	Miami Heat	98	0	19,6	1.00	3	3	0	1	1	0	0.34	10.210	-1.00	0.00	1.00	1
97	Dallas Mavericks	113	0	20,302	-1.00	4	6	2	0	0	0	0.00	2.650	0.00	0.00	1.00	1

Edat mitjana dels jugadors de cada equip

```
from statistics import mean
def age_avg(dim,players):
  team_capitals={'Toronto Raptors':'TOR', 'Boston Celtics':'BOS', 'Philadelphia 76ers':'PHI', 'Brooklyn Nets':'BRK', 'New York Knicks':'NYK','Milwaukee Bucks':'MIL', 'Indiana Pacers

  age={}
  for i in range(dim):
    if players['Tm'][i] not in age:
      age[players['Tm'][i]]=[players['Age'][i]]
    else:
      age[players['Tm'][i]].append(players['Age'][i])

  data['Age_visitor'] = data.apply(lambda row: sum(age[team_capitals[row.Visitor]])/len(age[team_capitals[row.Visitor]]) ,axis=1)
  data['Age_home'] = data.apply(lambda row: sum(age[team_capitals[row.Home]])/len(age[team_capitals[row.Home]]) ,axis=1)
  data['Age']=data.apply(lambda row: row.Age_home-row.Age_visitor, axis=1)
  return age
```

```
edat=age_avg(len(data_players),data_players)

data
```

tend.	Back to back	Div_Visitor	Div_Home	Division	Conference_Visitor	Conference_Home	Conference	Last N	Net Rating	OT last match	Rest/Injuries Visitor Stars	Rest/Injuries Home Stars	Rest/Injuries Stars	Win Streak	Km	Age_visitor
0,787	0.00	6	1	-5	0	1	1	0.31	0.000	0.00	0.00	0.00	0.00	0.00	-1789.00	24.875000
9,068	0.00	5	5	0	0	0	0	0.14	0.000	0.00	0.00	1.00	1.00	0.00	0.00	28.357143
5,424	0.00	2	3	1	1	1	0	0.21	0.000	0.00	0.00	0.00	0.00	0.00	-947.10	24.176471
7,923	0.00	2	2	0	1	1	0	0.09	0.000	0.00	1.00	1.00	0.00	0.00	-403.40	24.571429
8,846	0.00	2	3	1	1	1	0	0.28	0.000	0.00	0.00	0.00	0.00	0.00	-1436.00	24.307692
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
8,997	0.00	1	5	4	1	0	-1	0.45	8.400	0.00	1.00	0.00	-1.00	0.00	-1688.56	25.526316
0,172	1.00	2	1	-1	1	1	0	0.33	5.230	0.00	1.00	1.00	0.00	0.00	4296.74	24.571429
5,393	1.00	1	3	2	1	1	0	0.11	-1.410	1.00	0.00	0.00	0.00	0.00	229.58	24.533333
19,6	1.00	3	3	0	1	1	0	0.34	10.210	-1.00	0.00	1.00	1.00	0.00	1506.34	24.307692
0,302	-1.00	4	6	2	0	0	0	0.00	2.650	0.00	0.00	1.00	1.00	0.00	-4856.11	25.583333

Paràmetre que indica l'equip guanyador de cada partit

Aquesta funció ens retorna el paràmetre que indicarà si el partit l'ha guanyat l'equip local que es retornarà un 1 i si guanya l'equip visitant un 0.

Crearem una array 'winner' on guardarem els valors del guanyador de cada partit.

La funció transcorre de la següent manera:

1. Creem un loop amb for i in range( ) de la longitud del dataframe data d'aquesta manera analitzem cada fila que representa un partit de la temporada.
2. Per cada partit i equip guardem el valor associat, 1 si guanya l'equip local i 0 si guanya l'equip visitant.

```
data['Winner'] = data.apply(lambda row: 0 if row.PTSV>row.PTSH else 1 ,axis=1)
```

```
data
```

ack to ack	Div_Visitor	Div_Home	Division	Conference_Visitor	Conference_Home	Conference	Last N	Net Rating	OT last match	Rest/Injuries Visitor Stars	Rest/Injuries Home Stars	Rest/Injuries Stars	Win Streak	Km	Age_visitor	Age_hom
.00	6	1	-5	0	1	1	0.31	0.000	0.00	0.00	0.00	0.00	0.00	-1789.00	24.875000	25.4705
.00	5	5	0	0	0	0	0.14	0.000	0.00	0.00	1.00	1.00	0.00	0.00	28.357143	26.3571
.00	2	3	1	1	1	0	0.21	0.000	0.00	0.00	0.00	0.00	0.00	-947.10	24.176471	24.3076
.00	2	2	0	1	1	0	0.09	0.000	0.00	1.00	1.00	0.00	0.00	-403.40	24.571429	25.1250
.00	2	3	1	1	1	0	0.28	0.000	0.00	0.00	0.00	0.00	0.00	-1436.00	24.307692	25.8823
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
.00	1	5	4	1	0	-1	0.45	8.400	0.00	1.00	0.00	-1.00	0.00	-1688.56	25.526316	28.3571
.00	2	1	-1	1	1	0	0.33	5.230	0.00	1.00	1.00	0.00	0.00	4296.74	24.571429	25.6666
.00	1	3	2	1	1	0	0.11	-1.410	1.00	0.00	0.00	0.00	0.00	229.58	24.533333	25.1428
.00	3	3	0	1	1	0	0.34	10.210	-1.00	0.00	1.00	1.00	0.00	1506.34	24.307692	26.2941
.00	4	6	2	0	0	0	0.00	2.650	0.00	0.00	1.00	1.00	0.00	-4856.11	25.583333	27.0000

```
# Guardem a un fitxer .csv totes les dades processades per anar a la fase següent de anàlisi.
from google.colab import files
data.to_csv('NBA_analysis_19_20.csv')
files.download('NBA_analysis_19_20.csv')
```

Annex 2

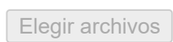
Fase IV. Modeling. Modelat

En aquesta fase, es seleccionen y s'utilitzen les tècniqeus de modelatge que siguin pertinents al problema (com més s'utilitzin millor), i es calibreen els seus paràmetres a valors òptims. Típicament hi ha diverses tècniques per el mateix problema de mineria de dades. Algunes tècniques tenen requeriments específics sobre la forma de les dades. Per tant casi sempre en qualsevol projecte s'acaba tornant a la fase de preparació de dades.

```
#Importem totes les llibreries requerides.

import pandas as pd
import pylab as pl
import numpy as np
import scipy.optimize as opt
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.metrics import jaccard_score
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import log_loss
import itertools
import seaborn as sns
import matplotlib as mpl
import matplotlib.pyplot as plt
%matplotlib inline
mpl.style.use('ggplot') # optional: for ggplot-like style
```

Obtenim les dades de la fase de preperació da dades que es troben el següent fitxer **.csv**.

```
from google.colab import files
uploaded = files.upload()
```


Elegir archivos Ningún archivo seleccionado Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving NBA_analysis_19_20.csv to NBA_analysis_19_20 (1).csv

```
import io
dataframe = pd.read_csv(io.BytesIO(uploaded['NBA_analysis_19_20.csv']))

dataframe=dataframe[['Date','Visitor','PTSV','Home','PTSH','Overtime','Attend.','%W','%W as Home/Visitor','Back to back','Div_Visitor','Div_Home','Division','Cor

dataframe.head()
```

	Date	Visitor	PTSV	Home	PTSH	Overtime	Attend.	%W	%W as Home/Visitor	Back to back	Div_Visitor	Div_Home	Division	Conference_Visitor	Conference_Home	Conference	Last N	Net Rating	OT last match
0	2019-10-22	New Orleans Pelicans	122	Toronto Raptors	130	1	20,787	0.073	0.176	0.0	6	1	-5	0	1	1	0.07	0.0	0.0
1	2019-10-22	Los Angeles Lakers	102	Los Angeles Clippers	112	0	19,068	0.049	0.180	0.0	5	5	0	0	0	0	0.05	0.0	0.0
2	2019-10-23	Chicago Bulls	125	Charlotte Hornets	126	0	15,424	-0.079	-0.013	0.0	2	3	1	1	1	0	-0.08	0.0	0.0
4	[illegible]	[illegible]	85	[illegible]	94	0	18,846	0.201	0.280	0.0	2	3	1	1	1	0	0.20	0.0	0.0

Quines variables ens donaran la millor predicció?

```
X=np.asarray(dataframe[['%W','%W as Home/Visitor','Back to back','Div_Visitor','Div_Home','Division',
                        'Conference_Visitor','Conference_Home','Conference','Last N','Net Rating','OT last match','Rest/Injuries Visitor Stars',
                        'Rest/Injuries Home Stars','Rest/Injuries Stars',
                        'Win Streak','Km','Age_visitor','Age_home','Age']])
y = np.asarray(dataframe['Winner'])


X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.31, random_state=5)
```

```
print ('Train set:', X_train.shape,  y_train.shape)
print ('Test set:', X_test.shape,  y_test.shape)

LR = LogisticRegression(C=0.9, solver='liblinear').fit(X_train,y_train)
yhat = LR.predict(X_test)
yhat_prob = LR.predict_proba(X_test)

jac=jaccard_score(y_test, yhat)

selector = RFE(LR, 1, step=1)
selector = selector.fit(X, y)
support=selector.support_
ranking=selector.ranking_


features_to_select = pd.DataFrame(columns=('Features','Support','Ranking'))
features_to_select['Features']=np.array(['%W','%W as Home/Visitor','Back to back','Div_Visitor','Div_Home','Division',
                        'Conference_Visitor','Conference_Home','Conference','Last N','Net Rating','OT last match','Rest/Injuries Visitor Stars',
                        'Rest/Injuries Home Stars','Rest/Injuries Stars',
                        'Win Streak','Km','Age_visitor','Age_home','Age'])
features_to_select['Support']=support
features_to_select['Ranking']=ranking
features=features_to_select.sort_values(by=['Ranking'])
features
```

```
Train set: (669, 20) (669,)
Test set: (302, 20) (302,)
```

	Features	Support	Ranking
0	%W	True	1
1	%W as Home/Visitor	False	2
11	OT last match	False	3
6	Conference_Visitor	False	4
9	Last N	False	5
19	Age	False	6
12	Rest/Injuries Visitor Stars	False	7
8	Conference	False	8
5	Division	False	9
7	Conference_Home	False	10
14	Rest/Injuries Stars	False	11
2	Back to back	False	12
18	Age_home	False	13
17	Age_visitor	False	14
4	Div_Home	False	15
3	Div_Visitor	False	16
10	Net Rating	False	17
15	Win Streak	False	18
13	Rest/Injuries Home Stars	False	19

Escollim les variables independents (X) que estan en un rang millor classificades en l'anàlisi RFE que hem realitzat anteriorment. També afegim la variable dependent (y) per el nostre model.

```
X = np.asarray(dataframe[['%W','%W as Home/Visitor','Conference_Visitor','OT last match','Last N','Age']])
X[0:5]
```

```
array([[ 0.07,  0.18,  0.  ,  0.  ,  0.07,  0.6 ],
       [ 0.05,  0.18,  0.  ,  0.  ,  0.05, -2.  ],
       [-0.08, -0.01,  1.  ,  0.  , -0.08,  0.13],
       [ 0.13,  0.24,  1.  ,  0.  ,  0.13,  0.55],
       [ 0.2 ,  0.28,  1.  ,  0.  ,  0.2 ,  1.57]])
```

```
y = np.asarray(dataframe['Winner'])
y [0:5]
```

```
array([1, 1, 1, 0, 1])
```

```
X = preprocessing.StandardScaler().fit(X).transform(X)
```

```
X = preprocessing.StandardScaler().fit(X).transform(X)
X[0:5]
```

```
array([[ 0.31,  0.16, -1.01,  0.02,  0.27,  0.31],
       [ 0.21,  0.17, -1.01,  0.02,  0.2 , -0.95],
       [-0.29, -0.49,  0.99,  0.02, -0.22,  0.08],
       [ 0.55,  0.38,  0.99,  0.02,  0.46,  0.29],
       [ 0.81,  0.52,  0.99,  0.02,  0.69,  0.78]])
```

Aquesta funció genera el Plot de la Confusion Matrixion matrix (aquest funció està extreta de un curset de machine learning no està programada per mi)

```
def plot_confusion_matrix(cm, classes,normalize=False,title='Confusion matrix',cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')
    print(cm)
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)
    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    print(confusion_matrix(y_test, yhat, labels=[1,0]))
    conf=(confusion_matrix(y_test, yhat, labels=[1,0]))
    conf=conf[0][0]+conf[1][1]
    print(conf)
    return conf
```

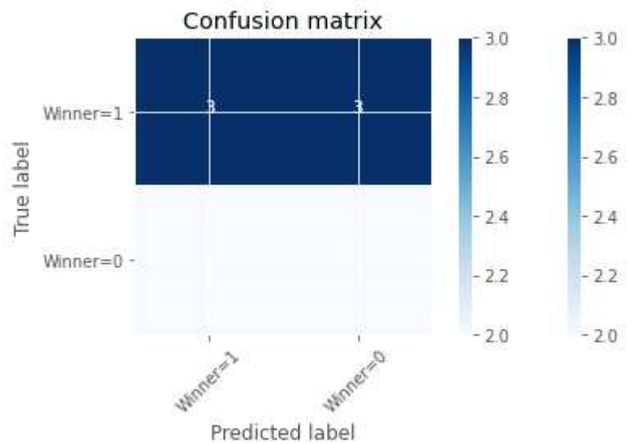
Anàlisi dels resultats a partir de cross validation of time series

Volem veure si podem veure com varia la precisió en la predicció al llarg de la temporada. Teòricament es podria esperar que la precisió augmentés a mesura que avança la temporada, però caldrà comprovar-ho amb resultats.

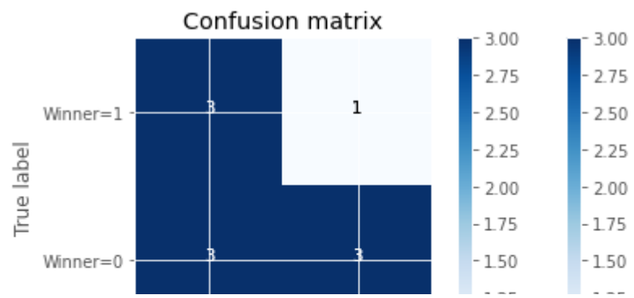
Per tal de realitzar aquest anàlisis, utilitzarem la cross validation of time series.

```
from sklearn.model_selection import TimeSeriesSplit

X = np.asarray(dataframe[['%W','%W as Home/Visitor','Conference_Visitor','OT last match','Last N','Age']])
y = np.asarray(dataframe['Winner'])

X = preprocessing.StandardScaler().fit(X).transform(X)

tscv = TimeSeriesSplit(n_splits=90)

correct=[]

for train_index, test_index in tscv.split(y):
    #print("%s %s" % (train_index, test_index))
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
    LR = LogisticRegression(C=0.1, solver='liblinear').fit(X_train,y_train)
    yhat = LR.predict(X_test)
    yhat_prob = LR.predict_proba(X_test)

    jaccard_score(y_test, yhat)
    # Compute confusion matrix
    cnf_matrix = confusion_matrix(y_test, yhat, labels=[1,0])
    np.set_printoptions(precision=2)

    # Plot non-normalized confusion matrix
    plt.figure()
    plot_confusion_matrix(cnf_matrix, classes=['Winner=1','Winner=0'],normalize= False,  title='Confusion matrix')
    cor=plot_confusion_matrix(cnf_matrix, classes=['Winner=1','Winner=0'],normalize= False,  title='Confusion matrix')
    correct=correct+[cor/float(len(y_test))]
    plt.show()

    print (classification_report(y_test, yhat))

    log_loss(y_test, yhat_prob)

    LR2 = LogisticRegression(C=0.1, solver='liblinear').fit(X_train,y_train)
    yhat_prob2 = LR2.predict_proba(X_test)
    print ("LogLoss: : %.2f" % log_loss(y_test, yhat_prob2))
```

```
Confusion matrix, without normalization
[[3 3]
 [2 2]]
[[3 3]
 [2 2]]
5
Confusion matrix, without normalization
[[3 3]
 [2 2]]
[[3 3]
 [2 2]]
5
```

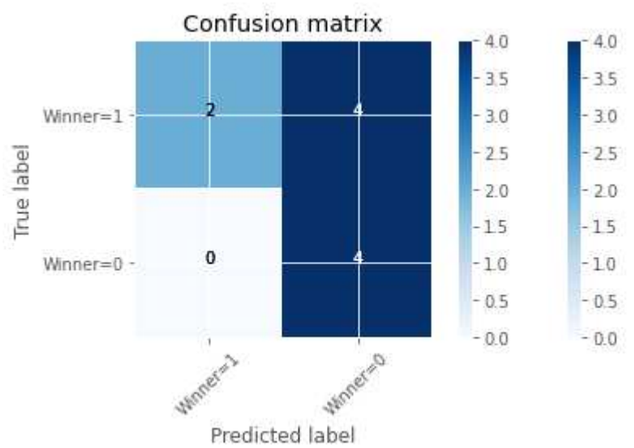



```
              precision    recall  f1-score   support

           0       0.40      0.50      0.44         4
           1       0.60      0.50      0.55         6

    accuracy                           0.50        10
   macro avg       0.50      0.50      0.49        10
weighted avg       0.52      0.50      0.51        10

LogLoss: : 0.63
Confusion matrix, without normalization
[[3 1]
 [3 3]]
[[3 1]
 [3 3]]
6
Confusion matrix, without normalization
[[3 1]
 [3 3]]
[[3 1]
 [3 3]]
6
```



	precision	recall	f1-score	support
0	0.75	0.50	0.60	6
1	0.50	0.75	0.60	4
accuracy			0.60	10
macro avg	0.62	0.62	0.60	10
weighted avg	0.65	0.60	0.60	10

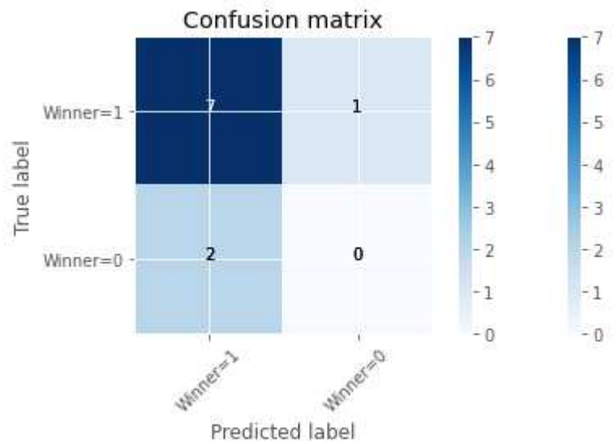
```
LogLoss: : 0.68
Confusion matrix, without normalization
[[2 4]
 [0 4]]
[[2 4]
 [0 4]]
6
Confusion matrix, without normalization
[[2 4]
 [0 4]]
[[2 4]
 [0 4]]
6
```



	precision	recall	f1-score	support
0	0.50	1.00	0.67	4
1	1.00	0.33	0.50	6
accuracy			0.60	10
macro avg	0.75	0.67	0.58	10
weighted avg	0.80	0.60	0.57	10

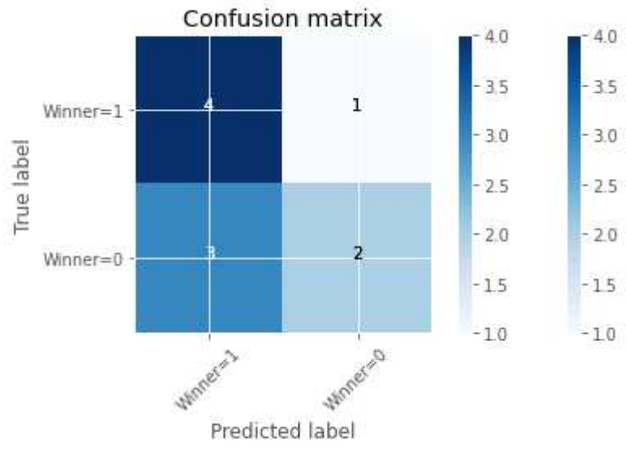
```
LogLoss: : 0.61
Confusion matrix, without normalization
[[7 1]
 [2 0]]
[[7 1]
 [2 0]]
7
Confusion matrix, without normalization
```

```
[[7 1]
 [2 0]]
[[7 1]
 [2 0]]
7
```



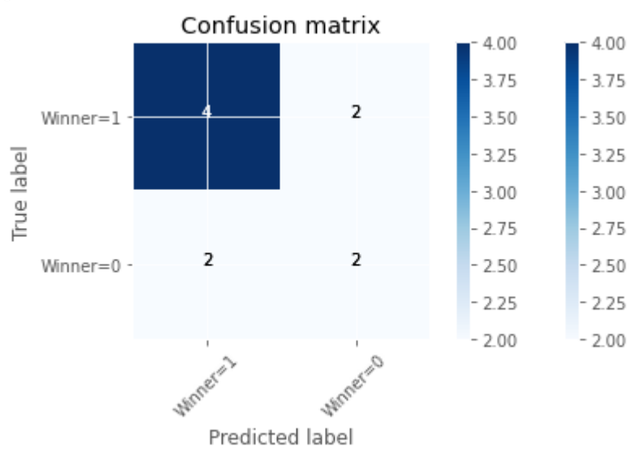
	precision	recall	f1-score	support
0	0.00	0.00	0.00	2
1	0.78	0.88	0.82	8
accuracy			0.70	10
macro avg	0.39	0.44	0.41	10
weighted avg	0.62	0.70	0.66	10

```
LogLoss: : 0.56
Confusion matrix, without normalization
[[4 1]
 [3 2]]
[[4 1]
 [3 2]]
6
Confusion matrix, without normalization
[[4 1]
 [3 2]]
[[4 1]
 [3 2]]
6
```

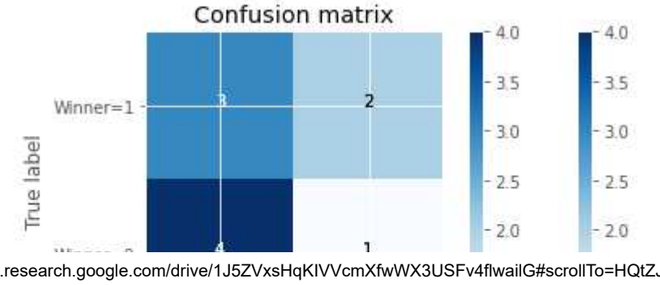


	precision	recall	f1-score	support

```
LogLoss: : 0.84
Confusion matrix, without normalization
[[4 2]
 [2 2]]
[[4 2]
 [2 2]]
6
Confusion matrix, without normalization
[[4 2]
 [2 2]]
[[4 2]
 [2 2]]
6
```




```
              precision    recall  f1-score   support

           0       0.50      0.50      0.50         4
           1       0.67      0.67      0.67         6

    accuracy                           0.60        10
   macro avg       0.58      0.58      0.58        10
weighted avg       0.60      0.60      0.60        10

LogLoss: : 0.67
Confusion matrix, without normalization
[[3 2]
 [4 1]]
[[3 2]
 [4 1]]
4
Confusion matrix, without normalization
[[3 2]
 [4 1]]
[[3 2]
 [4 1]]
4
```




```
correct
l=np.arange(len(correct))
l
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
       34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
       51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67,
       68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84,
       85, 86, 87, 88, 89])
```

```
LogLoss: : 0.87
```

```
corr=pd.DataFrame()
#corr['Split'] = l
corr['Average Correct']=correct
corr
```

	Average Correct
0	0.5
1	0.6
2	0.6
3	0.7
4	0.6
...	...
85	0.7
86	0.5
87	0.4
88	0.6
89	0.4

90 rows × 1 columns

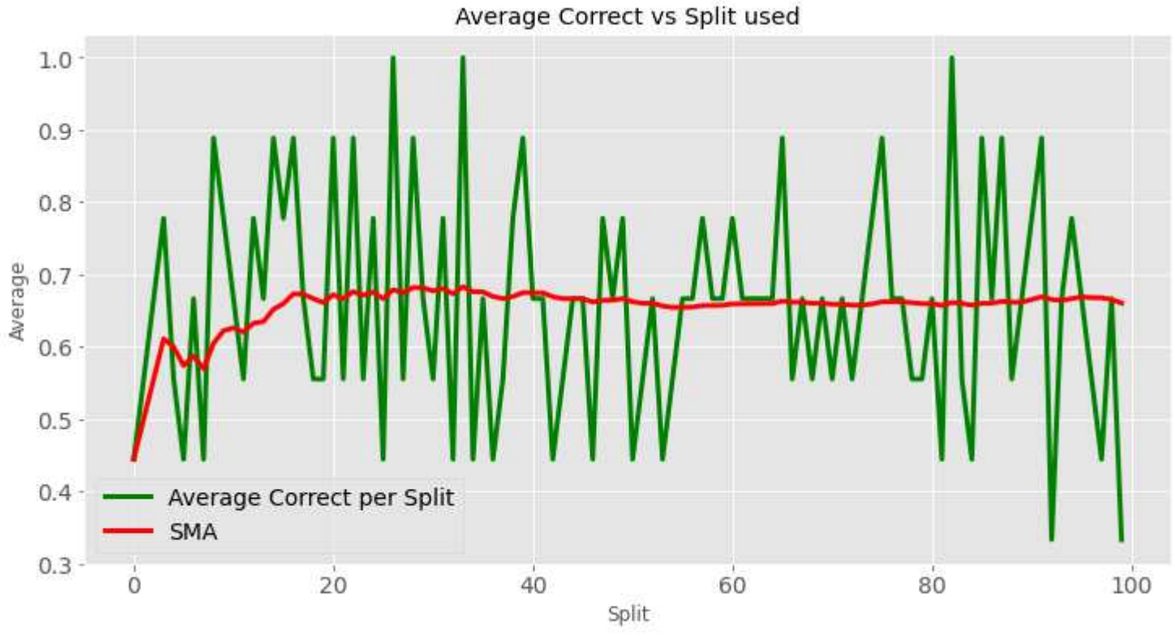
```
corr['SMA'] = corr['Average Correct'].rolling(90, min_periods=1).mean()
```

```
corr
```

	Average Correct	SMA
0	0.444444	0.444444
1	0.555556	0.500000
2	0.666667	0.555556
3	0.777778	0.611111
4	0.555556	0.600000
...	...	...
95	0.666667	0.669136
96	0.555556	0.667901

```
#corr.plot(kind='line', y='Average Correct', figsize=(15,6), color='darkblue')
colors = ['green', 'red']
corr.plot(color=colors, linewidth=3, figsize=(12,6))

#corr.plot.line(x='Split', y='Average Correct')
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.legend(labels =['Average Correct per Split','SMA'], fontsize=14)
plt.title('Average Correct vs Split used')
plt.xlabel('Split')
plt.ylabel('Average')
```

```
Text(0, 0.5, 'Average')
```




```
corr['Average Correct'].describe()
```

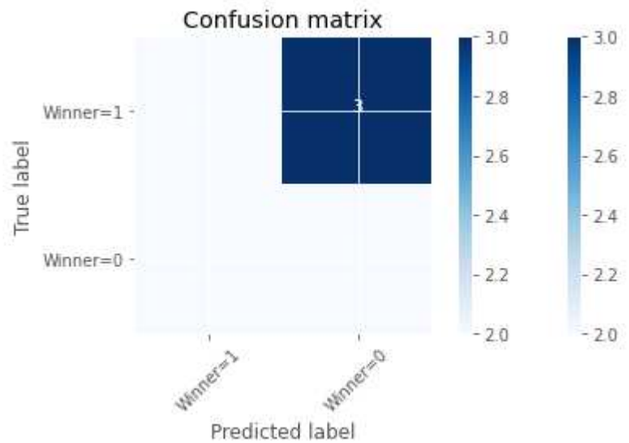
```
count    100.000000
mean       0.656667
```

```
std        0.151556
min        0.333333
25%        0.555556
50%        0.666667
75%        0.777778
max        1.000000
Name: Average Correct, dtype: float64
```

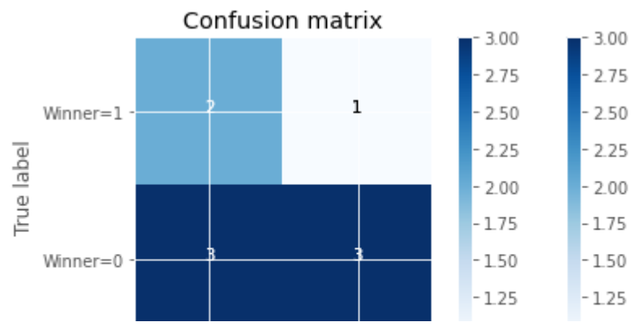
Anàlisi a partir de l'algoritme de classificació SVM

```
from sklearn import svm
from sklearn.model_selection import TimeSeriesSplit


X = np.asarray(dataframe[['%W','%W as Home/Visitor','Conference_Visitor','OT last match','Last N','Age']])
y = np.asarray(dataframe['Winner'])


X = preprocessing.StandardScaler().fit(X).transform(X)

tscv = TimeSeriesSplit(n_splits=100)

correct=[]

for train_index, test_index in tscv.split(y):
    #print("%s %s" % (train_index, test_index))
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]


    clf = svm.SVC(kernel='linear',C=0.1)
    clf.fit(X_train, y_train)
    yhat = clf.predict(X_test)

    jaccard_score(y_test, yhat)

    # Compute confusion matrix
    cnf_matrix = confusion_matrix(y_test, yhat, labels=[1,0])
    np.set_printoptions(precision=2)

    # Plot non-normalized confusion matrix
    plt.figure()
    plot_confusion_matrix(cnf_matrix, classes=['Winner=1','Winner=0'],normalize= False,  title='Confusion matrix')
    cor=plot_confusion_matrix(cnf_matrix, classes=['Winner=1','Winner=0'],normalize= False,  title='Confusion matrix')
    correct=correct+[cor/float(len(y_test))]
    plt.show()

    print (classification_report(y_test, yhat))
```

```
Confusion matrix, without normalization
[[2 3]
 [2 2]]
[[2 3]
 [2 2]]
4
Confusion matrix, without normalization
[[2 3]
 [2 2]]
[[2 3]
 [2 2]]
4
```

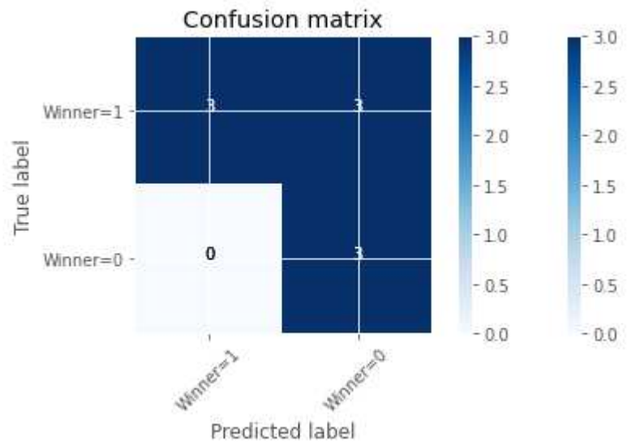



```
              precision    recall  f1-score   support

           0       0.40      0.50      0.44         4
           1       0.50      0.40      0.44         5

    accuracy                           0.44         9
   macro avg       0.45      0.45      0.44         9
weighted avg       0.46      0.44      0.44         9

Confusion matrix, without normalization
[[2 1]
 [3 3]]
[[2 1]
 [3 3]]
5
Confusion matrix, without normalization
[[2 1]
 [3 3]]
[[2 1]
 [3 3]]
5
```



	precision	recall	f1-score	support
0	0.75	0.50	0.60	6
1	0.40	0.67	0.50	3
accuracy			0.56	9
macro avg	0.57	0.58	0.55	9
weighted avg	0.63	0.56	0.57	9

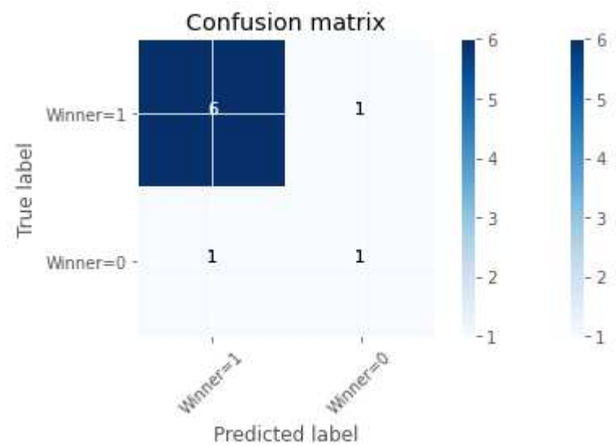
```
Confusion matrix, without normalization
[[3 3]
 [0 3]]
[[3 3]
 [0 3]]
6
Confusion matrix, without normalization
[[3 3]
 [0 3]]
[[3 3]
 [0 3]]
6
```



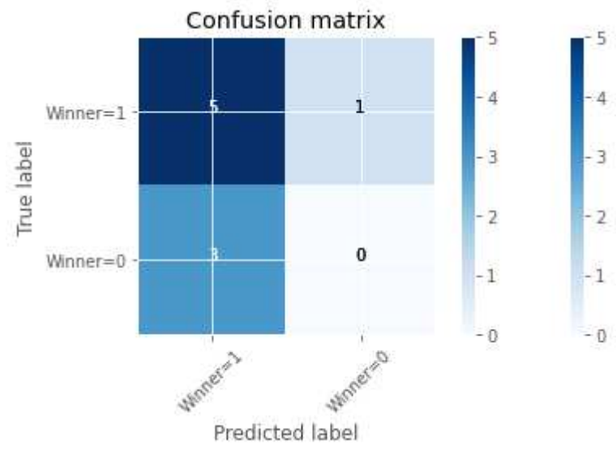
	precision	recall	f1-score	support
0	0.50	1.00	0.67	3
1	1.00	0.50	0.67	6
accuracy			0.67	9
macro avg	0.75	0.75	0.67	9
weighted avg	0.83	0.67	0.67	9

```
Confusion matrix, without normalization
[[6 1]
 [1 1]]
[[6 1]
 [1 1]]
7
Confusion matrix, without normalization
[[6 1]
 [1 1]]
[[6 1]
```

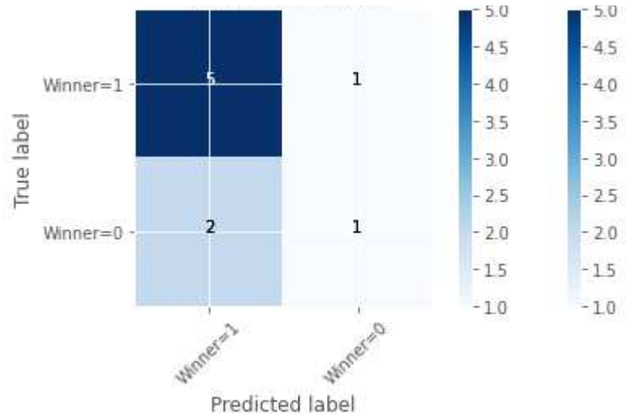
```
 [1 1]]
7
```




```
              precision    recall  f1-score   support

           0       0.50      0.50      0.50         2
           1       0.86      0.86      0.86         7

    accuracy                           0.78         9
   macro avg       0.68      0.68      0.68         9
weighted avg       0.78      0.78      0.78         9

Confusion matrix, without normalization
[[5 1]
 [3 0]]
5
Confusion matrix, without normalization
[[5 1]
 [3 0]]
[[5 1]
 [3 0]]
5
```

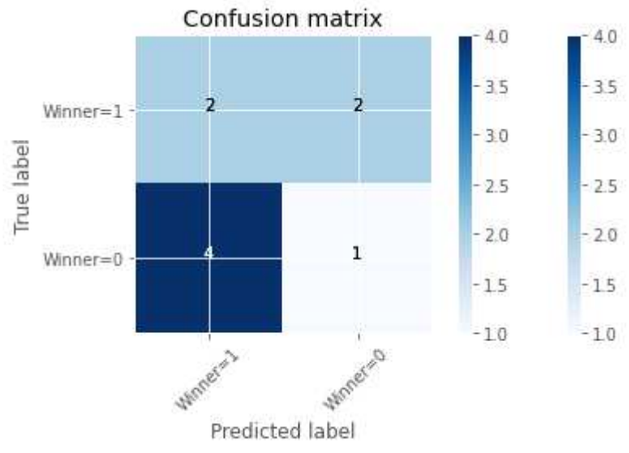



```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         3
           1       0.62      0.83      0.71         6

    accuracy                           0.56         9
```



	precision	recall	f1-score	support
0	0.50	0.33	0.40	3
1	0.71	0.83	0.77	6
accuracy			0.67	9
macro avg	0.61	0.58	0.58	9
weighted avg	0.64	0.67	0.65	9

```
Confusion matrix, without normalization
[[2 2]
 [4 1]]
[[2 2]
 [4 1]]
3
```



	precision	recall	f1-score	support
0	0.33	0.20	0.25	5
1	0.33	0.50	0.40	4
accuracy			0.33	9
macro avg	0.33	0.35	0.33	9
weighted avg	0.33	0.33	0.32	9

```
correct
l=np.arange(len(correct))
l
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
       34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
       51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67,
       68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84,
       85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99])
```

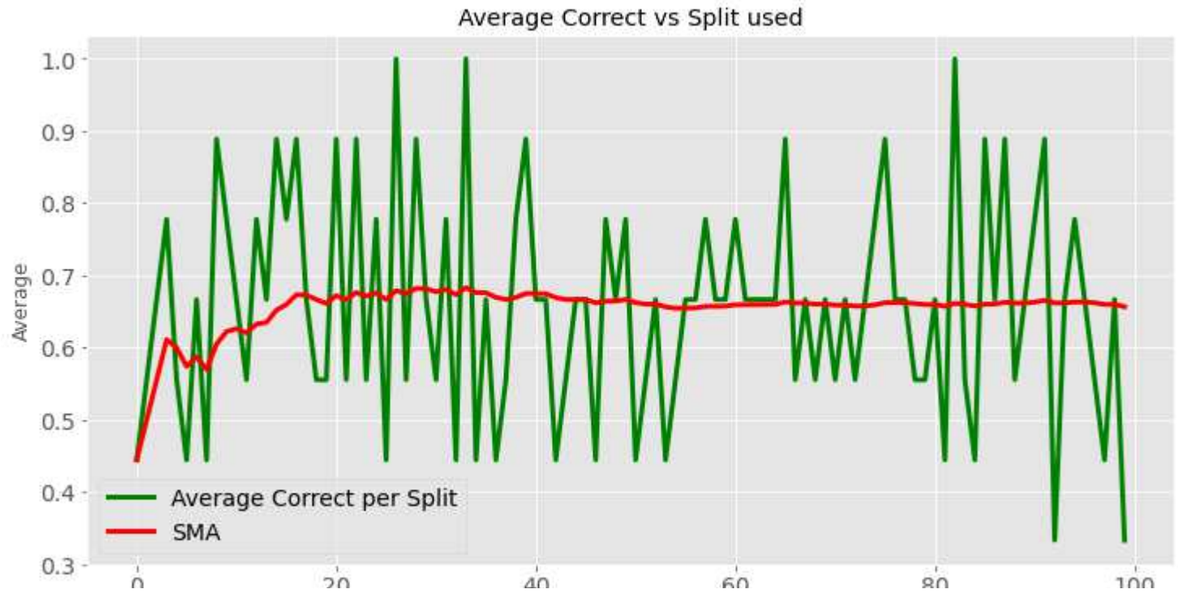
```
corr=pd.DataFrame()
#corr['Split'] = l
corr['Average Correct']=correct
corr['SMA'] = corr['Average Correct'].rolling(100, min_periods=1).mean()

corr
```

	Average Correct	SMA
0	0.444444	0.444444
1	0.555556	0.500000
2	0.666667	0.555556
3	0.777778	0.611111
4	0.555556	0.600000
...	...	...
95	0.666667	0.663194
96	0.555556	0.662085
97	0.444444	0.659864
98	0.666667	0.659933
99	0.333333	0.656667

100 rows × 2 columns

```
#corr.plot(kind='line', y='Average Correct', figsize=(15,6), color='darkblue')
colors = ['green', 'red']
corr.plot(color=colors, linewidth=3, figsize=(12,6))

#corr.plot.line(x='Split', y='Average Correct')
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.legend(labels =['Average Correct per Split','SMA'], fontsize=14)
plt.title('Average Correct vs Split used')
plt.xlabel('Split')
plt.ylabel('Average')
```

```
Text(0, 0.5, 'Average')
```




```
corr.describe()
```

	Average Correct	SMA
count	100.000000	100.000000
mean	0.656667	0.653380
std	0.151556	0.034855
min	0.333333	0.444444
25%	0.555556	0.657530
50%	0.666667	0.661492
75%	0.777778	0.666667
max	1.000000	0.683007

Time series aplicada per equips

```
TOR=dataframe.loc[(dataframe['Visitor'] == 'Toronto Raptors') | (dataframe['Home'] == 'Toronto Raptors')]
TOR
```

	Date	Visitor	PTSV	Home	PTSH	Overtime	Attend.	%W	%W as Home/Visitor	Back to back	Div_Visitor	Div_Home	Division	Conference_Visitor	Conference_Home	Conference	Last N	Net Rating	la mat
0	2019-10-22	New Orleans Pelicans	122	Toronto Raptors	130	1	20,787	0.073	0.176	0.0	6	1	-5	0	1	1	0.07	0.00	
16	2019-10-25	Toronto Raptors	106	Boston Celtics	112	0	18,624	0.055	0.170	-1.0	1	1	0	1	1	0	0.05	0.00	-
29	2019-10-26	Toronto Raptors	108	Chicago Bulls	84	0	21,498	-0.177	-0.085	0.0	1	2	1	1	1	0	-0.18	7.00	
42	2019-10-28	Orlando Magic	95	Toronto Raptors	104	0	19,8	0.055	0.159	0.0	3	1	-2	1	1	0	0.06	8.00	
59	2019-10-30	Detroit Pistons	113	Toronto Raptors	125	0	19,8	0.092	0.192	0.0	2	1	-1	1	1	0	0.09	12.00	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
896	2020-03-01	Toronto Raptors	118	Denver Nuggets	133	0	19,777	-0.034	0.063	0.0	1	4	3	1	0	-1	0.00	-3.42	
912	2020-03-03	Toronto Raptors	123	Phoenix Suns	114	0	15,553	-0.307	-0.335	1.0	1	5	4	1	0	-1	-0.23	-8.07	
927	2020-03-05	Toronto Raptors	121	Golden State Warriors	113	0	18,064	-0.479	-0.464	0.0	1	5	4	1	0	-1	-0.34	-14.65	

```
teams={'Toronto Raptors':'TOR', 'Boston Celtics':'BOS', 'Philadelphia 76ers':'PHI', 'Brooklyn Nets':'BRK', 'New York Knicks':'NYK',
       'Milwaukee Bucks':'MIL', 'Indiana Pacers':'IND', 'Chicago Bulls':'CHI','Detroit Pistons':'DET',
       'Miami Heat':'MIA', 'Orlando Magic':'ORL','Washington Wizards':'WAS','Charlotte Hornets':'CHO','Atlanta Hawks':'ATL','Denver Nuggets':'DEN',
       'Utah Jazz':'UTA','Oklahoma City Thunder':'OKC','Portland Trail Blazers':'POR', 'Minnesota Timberwolves':'MIN',
       'Los Angeles Clippers':'LAC','Sacramento Kings':'SAC','Phoenix Suns':'PHO','Golden State Warriors':'GSW','Houston Rockets':'HOU',
       'Dallas Mavericks':'DAL','Memphis Grizzlies':'MEM','New Orleans Pelicans':'NOP','Cleveland Cavaliers':'CLE','Los Angeles Lakers':'LAL','San Antonio Spurs'


for key in teams:
  teams[key]=dataframe.loc[(dataframe['Visitor'] == key) | (dataframe['Home'] == key)]

teams['Cleveland Cavaliers']
```

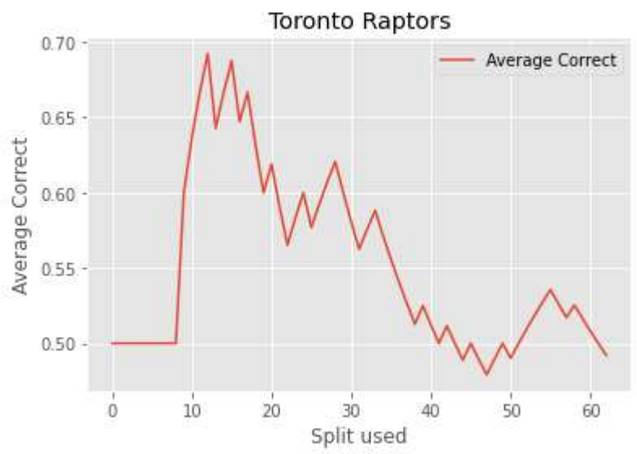
	Date	Visitor	PTSV	Home	PTSH	Overtime	Attend.	%W	%W as Home/Visitor	Back to back	Div_Visitor	Div_Home	Division	Conference_Visitor	Conference_Home	Conference	Last N	Net Rating	la mat
4	2019-10-23	Cleveland Cavaliers	85	Orlando Magic	94	0	18,846	0.201	0.280	0.0	2	3	1	1	1	0	0.20	0.00	
30	2019-10-26	Indiana Pacers	99	Cleveland Cavaliers	110	0	19,432	-0.298	-0.210	0.0	2	2	0	1	1	0	-0.30	0.00	
45	2019-10-28	Cleveland Cavaliers	112	Milwaukee Bucks	129	0	17,385	0.402	0.502	0.0	2	2	0	1	1	0	0.41	-8.00	
54	2019-10-30	Chicago Bulls	111	Cleveland Cavaliers	117	0	17,595	-0.079	-0.013	0.0	2	2	0	1	1	0	-0.08	3.75	
69	2019-11-01	Cleveland Cavaliers	95	Indiana Pacers	102	0	16,079	0.298	0.388	0.0	2	2	0	1	1	0	0.30	-0.75	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
900	2020-03-02	Utah Jazz	126	Cleveland Cavaliers	113	0	15,453	-0.344	-0.271	-1.0	4	2	-2	0	1	1	-0.12	-11.05	

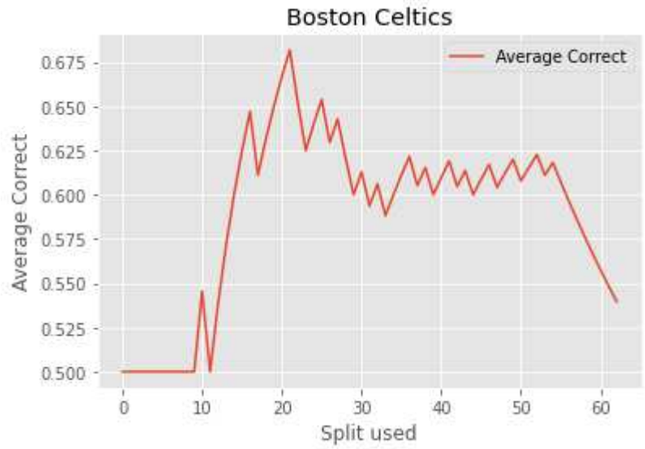
```
for key in teams:
  s=[]
  accuracy=[]
  X = np.asarray(teams[key][['%W','%W as Home/Visitor','Conference_Visitor','OT last match','Last N','Age']])
  y = np.asarray(teams[key]['Winner'])

  X = preprocessing.StandardScaler().fit(X).transform(X)

  tscv = TimeSeriesSplit(n_splits=len(X)-1)

  #correct=[]

  for train_index, test_index in tscv.split(y):
    #print("%s %s" % (train_index, test_index))
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    if len(X_test)>=2:
      clf = svm.SVC(kernel='linear',C=1)
      clf.fit(X_train, y_train)
      yhat = clf.predict(X_test)
    else:
      pass


    jaccard_score(y_test, yhat)
    s=s+[jaccard_score(y_test, yhat)]
    if len(s)<10:
      accuracy=accuracy+[0.5]
    else:
      accuracy=accuracy+[sum(s)/float(len(s))]
  print(key)
  print(len(s))
```

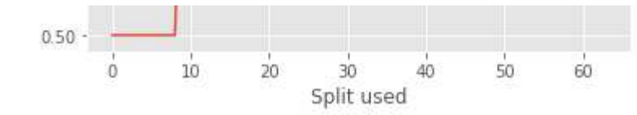
```
    print(s)
    print(sum(s))
    print(tscv)
    print(accuracy)
    s=[]
    l=np.arange(len(accuracy))
    corr=pd.DataFrame()
    corr['Split'] = l
    corr['Average Correct']=accuracy
    corr.plot.line(x='Split', y='Average Correct')
    plt.title(key)
    plt.xlabel('Split used')
    plt.ylabel('Average Correct')
```

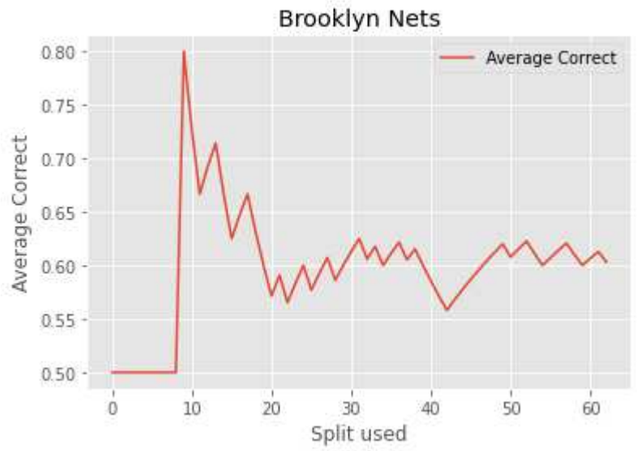
```
Toronto Raptors
63
[1.0, 0.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 1.0, 0.0, 1.0, 1.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.
31.0
TimeSeriesSplit(max_train_size=None, n_splits=63)
[0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.6, 0.6363636363636364, 0.6666666666666666, 0.6923076923076923, 0.6428571428571429, 0.6666666666666666, 0.6875, 0.6470588235294118, 0.666666
Boston Celtics
63
[1.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 1.0, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 1.0, 1.0, 0.
34.0
TimeSeriesSplit(max_train_size=None, n_splits=63)
[0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5454545454545454, 0.5, 0.5384615384615384, 0.5714285714285714, 0.6, 0.625, 0.6470588235294118, 0.6111111111111112, 0.631578947368421, 0
Philadelphia 76ers
64
[0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.
52.0
TimeSeriesSplit(max_train_size=None, n_splits=64)
[0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.7, 0.7272727272727273, 0.6666666666666666, 0.6923076923076923, 0.7142857142857143, 0.7333333333333333, 0.75, 0.7647058823529411, 0.72222222
Brooklyn Nets
63
[1.0, 1.0, 0.0, 1.0, 1.0, 1.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 0.0, 1.0, 1.0, 0.0, 1.0, 1.0, 1.0, 0.0, 1.0, 0.0, 1.0, 1.0, 0.
38.0
TimeSeriesSplit(max_train_size=None, n_splits=63)
[0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.8, 0.7272727272727273, 0.6666666666666666, 0.6923076923076923, 0.7142857142857143, 0.6666666666666666, 0.625, 0.6470588235294118, 0.6666666
New York Knicks
65
[1.0, 0.0, 1.0, 1.0, 1.0, 0.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.
33.0
TimeSeriesSplit(max_train_size=None, n_splits=65)
[0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.6, 0.6363636363636364, 0.5833333333333334, 0.6153846153846154, 0.6428571428571429, 0.6, 0.5625, 0.5882352941176471, 0.5555555555555556, 0.52
Milwaukee Bucks
64
[0.0, 1.0, 1.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 0.0, 1.0, 1.0, 0.0, 1.0, 1.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 1.0, 0.0, 1.0, 0.0, 1.0, 1.0, 1.0, 0.
37.0
TimeSeriesSplit(max_train_size=None, n_splits=64)
[0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.45454545454545453, 0.4166666666666667, 0.38461538461538464, 0.42857142857142855, 0.4666666666666667, 0.5, 0.5294117647058824, 0.5, 0.52
Indiana Pacers
64
[1.0, 1.0, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.
36.0
TimeSeriesSplit(max_train_size=None, n_splits=64)
[0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.8, 0.8181818181818182, 0.75, 0.6923076923076923, 0.7142857142857143, 0.7333333333333333, 0.75, 0.7647058823529411, 0.7777777777777778, 0.736
Chicago Bulls
64
[0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.
36.0
TimeSeriesSplit(max_train_size=None, n_splits=64)
[0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5454545454545454, 0.5, 0.46153846153846156, 0.5, 0.4666666666666667, 0.4375, 0.4117647058823529, 0.4444444444444444, 0.473684210526315
Detroit Pistons
65
[0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 0.0, 1.0, 0.0, 0.
35.0
TimeSeriesSplit(max_train_size=None, n_splits=65)
[0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.7, 0.7272727272727273, 0.75, 0.7692307692307693, 0.7857142857142857, 0.8, 0.8125, 0.8235294117647058, 0.7777777777777778, 0.7894736842105263
Miami Heat
64
[0.0, 1.0, 1.0, 0.0, 1.0, 1.0, 0.0, 1.0, 1.0, 0.0, 1.0, 1.0, 0.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 1.0, 1.
45.0
TimeSeriesSplit(max_train_size=None, n_splits=64)
[0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.6, 0.6363636363636364, 0.6666666666666666, 0.6153846153846154, 0.6428571428571429, 0.6666666666666666, 0.6875, 0.7058823529411765, 0.666666
Orlando Magic
```

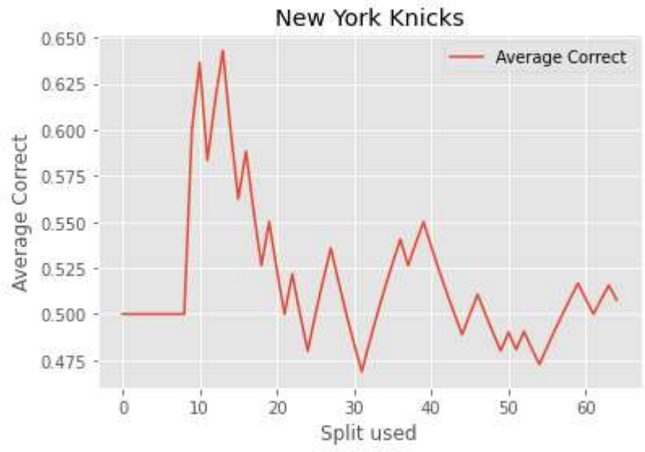
```
Los Angeles Lakers
62
[1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.
28.0
TimeSeriesSplit(max_train_size=None, n_splits=62)
[0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5454545454545454, 0.5833333333333334, 0.6153846153846154, 0.5714285714285714, 0.5333333333333333, 0.5, 0.47058823529411764, 0.5, 0.473
/usr/local/lib/python3.6/dist-packages/pandas/plotting/_matplotlib/core.py:328: RuntimeWarning: More than 20 figures have been opened. Figures created through the pyplot interface (`matplo
  fig = self.plt.figure(figsize=self.figsize)
/usr/local/lib/python3.6/dist-packages/pandas/plotting/_matplotlib/core.py:328: RuntimeWarning: More than 20 figures have been opened. Figures created through the pyplot interface (`matplo
  fig = self.plt.figure(figsize=self.figsize)
San Antonio Spurs
62
[1.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 1.0, 1.0, 0.0, 1.0, 0.0, 1.
37.0
TimeSeriesSplit(max_train_size=None, n_splits=62)
[0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.6, 0.6363636363636364, 0.5833333333333334, 0.6153846153846154, 0.6428571428571429, 0.6666666666666666, 0.625, 0.5882352941176471, 0.5555555
```

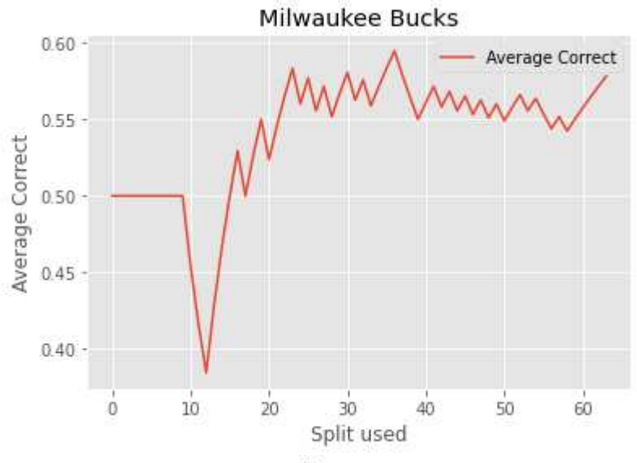




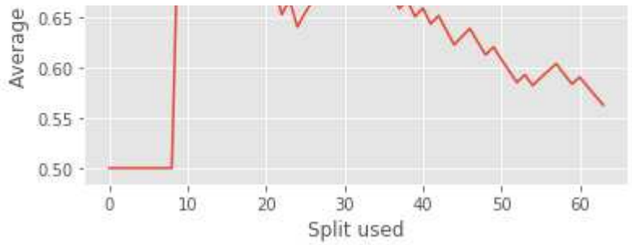

0.50
0
10
20
30
40
50
60
Split used


Brooklyn Nets
Average Correct
0.80
0.75
0.70
0.65
0.60
0.55
0.50
Average Correct
0
10
20
30
40
50
60
Split used

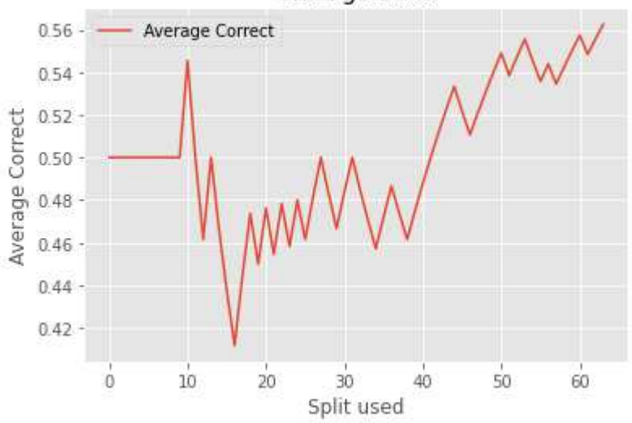

New York Knicks
Average Correct
0.650
0.625
0.600
0.575
0.550
0.525
0.500
0.475
Average Correct
0
10
20
30
40
50
60
Split used


Milwaukee Bucks
Average Correct
0.60
0.55
0.50
0.45
0.40
Average Correct
0
10
20
30
40
50
60
Split used

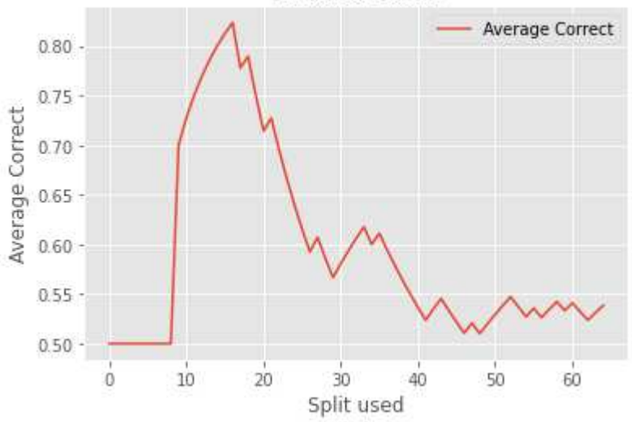

Indiana Pacers
Average Correct
0.80
0.75
0.70
Correct



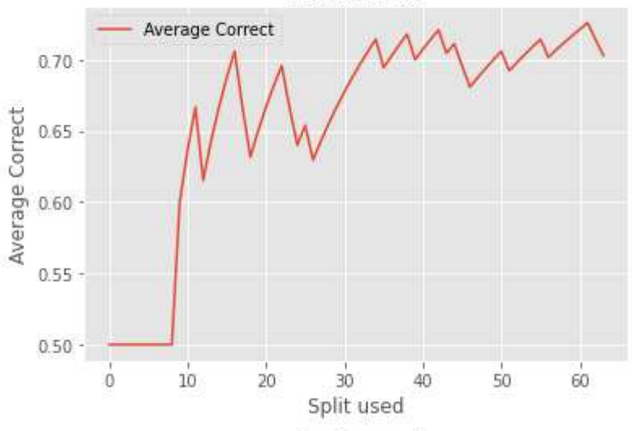
Chicago Bulls



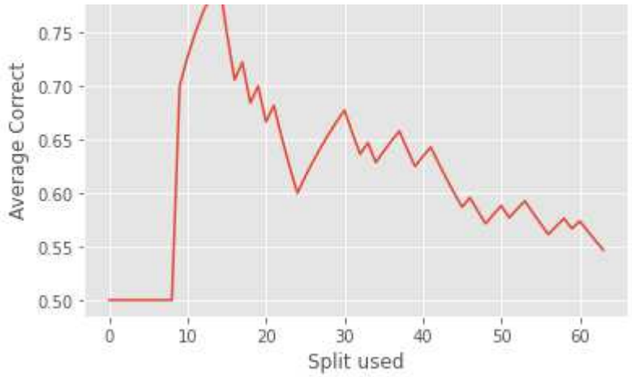
Detroit Pistons

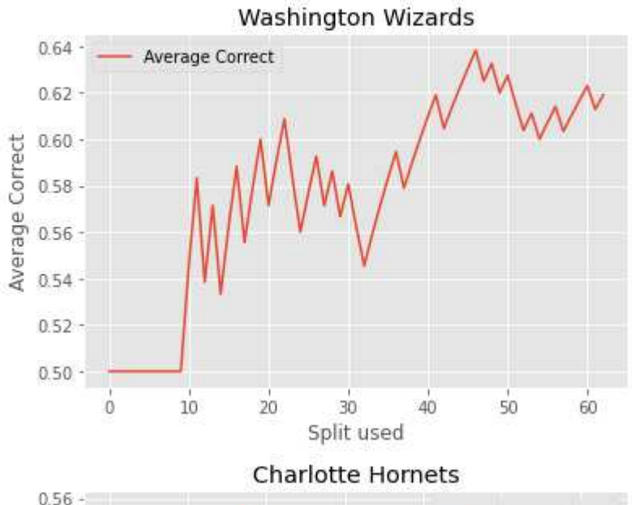


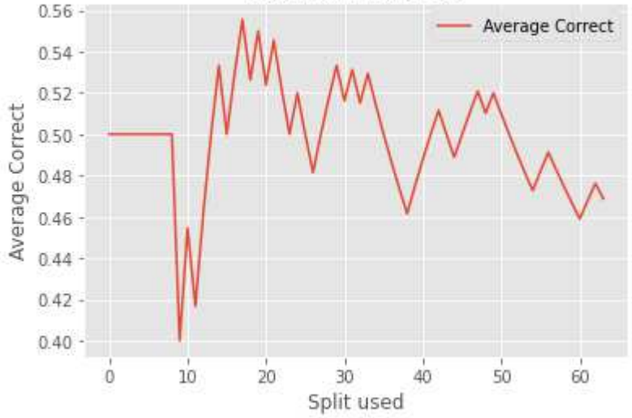
Miami Heat

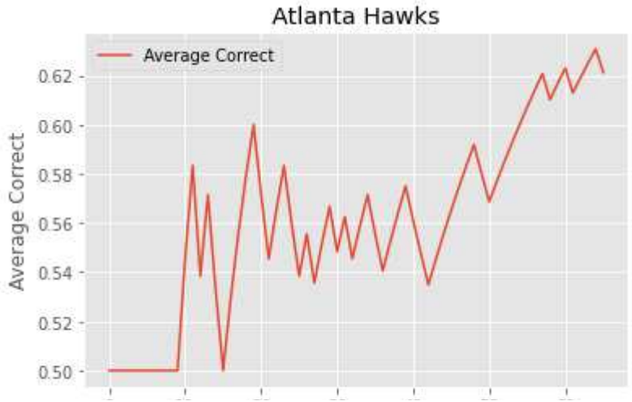


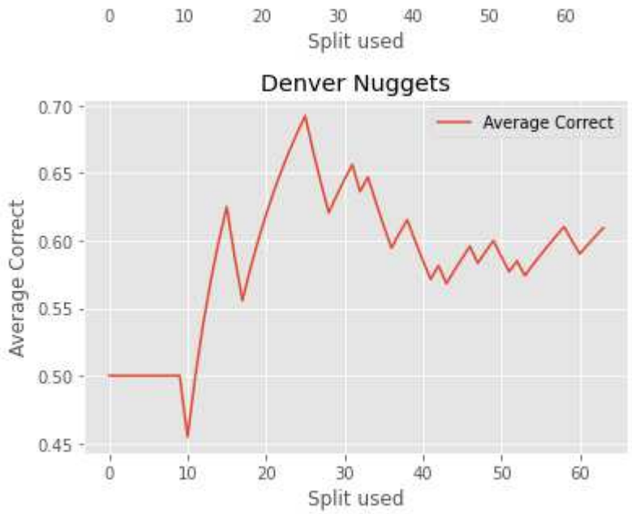
Orlando Magic

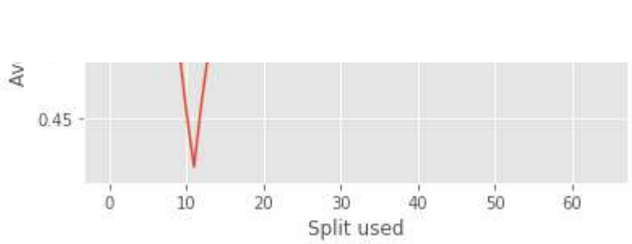

Average Correct
Split used

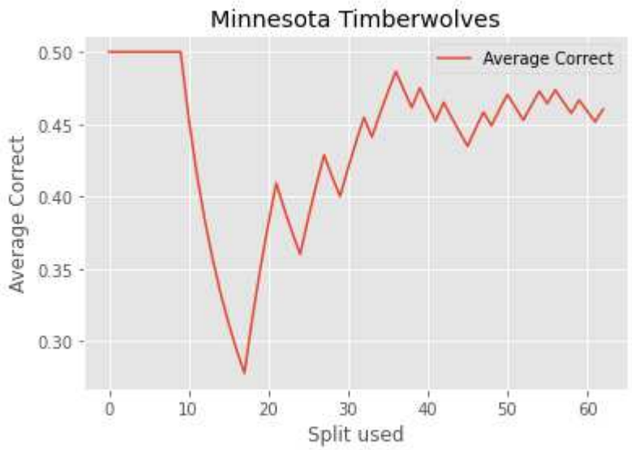

Washington Wizards
Average Correct
Average Correct
Split used

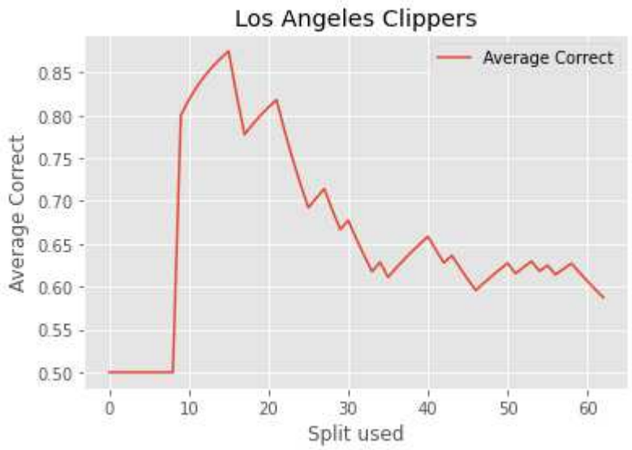

Charlotte Hornets
Average Correct
Average Correct
Split used

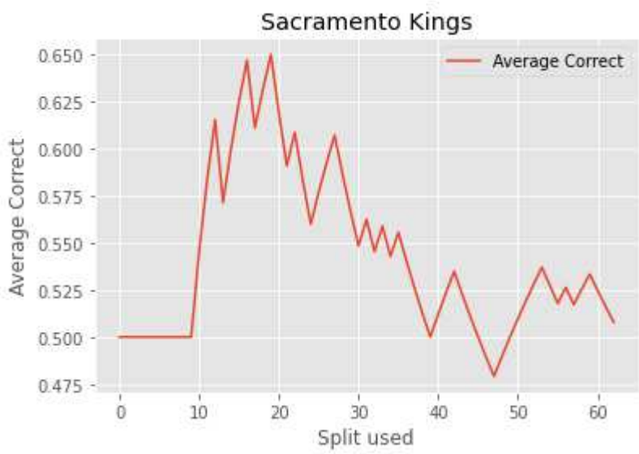

Atlanta Hawks
Average Correct
Average Correct


Split used
Denver Nuggets
Average Correct
Average Correct
Split used
Utah Jazz
Average Correct
Average Correct
Split used
Oklahoma City Thunder
Average Correct
Average Correct
Split used
Portland Trail Blazers
Average Correct

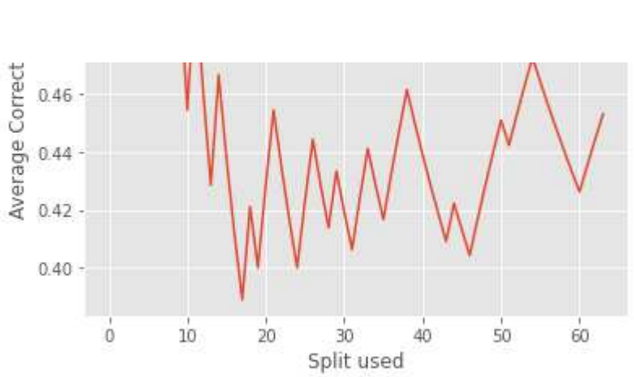

0.45
0
10
20
30
40
50
60
Split used


Minnesota Timberwolves
Average Correct
0.50
0.45
0.40
0.35
0.30
Average Correct
0
10
20
30
40
50
60
Split used

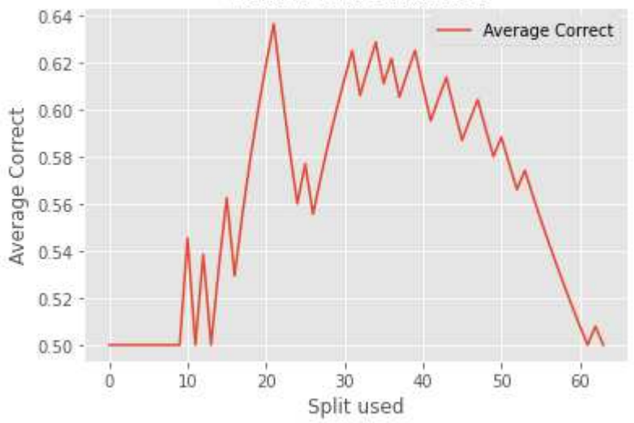

Los Angeles Clippers
Average Correct
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0.50
Average Correct
0
10
20
30
40
50
60
Split used


Sacramento Kings
Average Correct
0.650
0.625
0.600
0.575
0.550
0.525
0.500
0.475
Average Correct
0
10
20
30
40
50
60
Split used

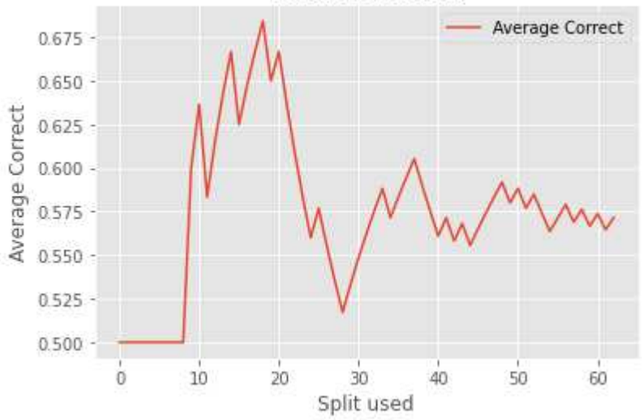
Phoenix Suns
Average Correct
0.50
0.48



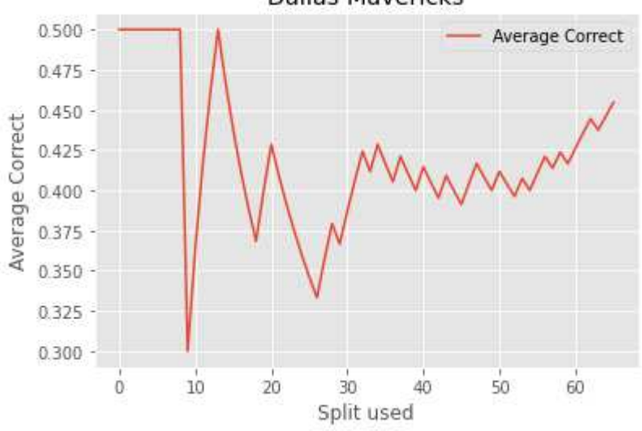
Golden State Warriors

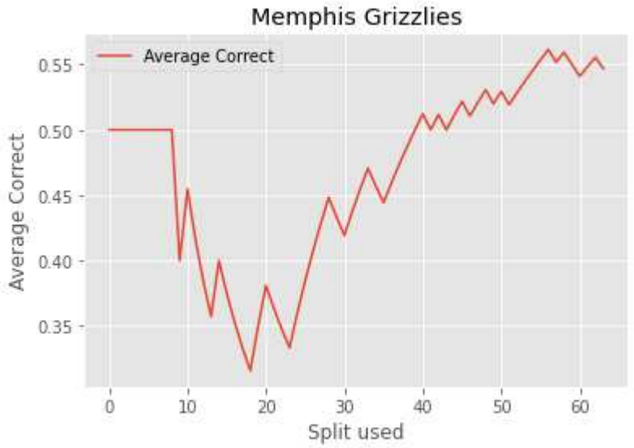


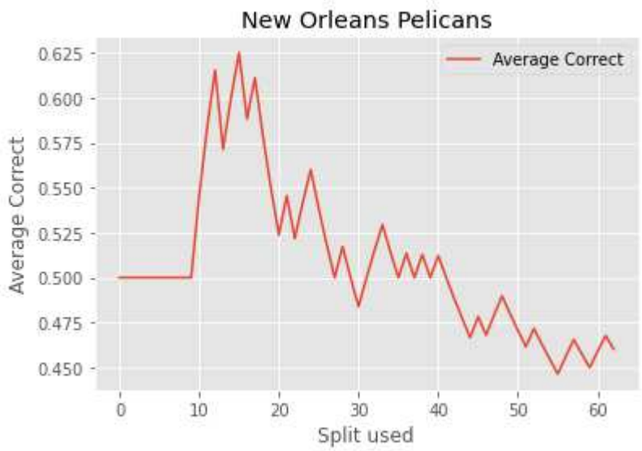
Houston Rockets

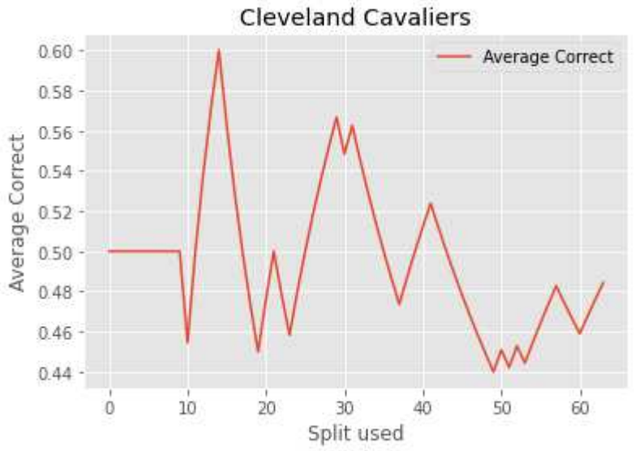


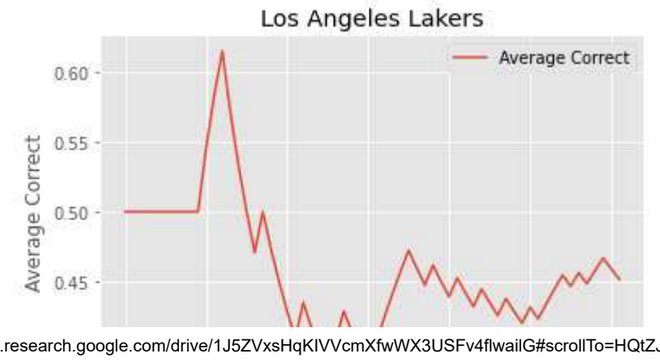
Dallas Mavericks

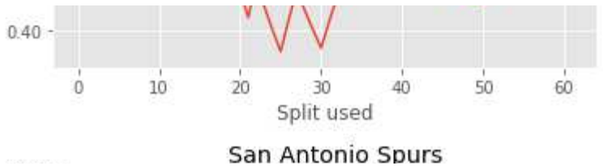



Memphis Grizzlies
Average Correct
Average Correct
Split used
0.55
0.50
0.45
0.40
0.35
0
10
20
30
40
50
60


New Orleans Pelicans
Average Correct
Average Correct
Split used
0.625
0.600
0.575
0.550
0.525
0.500
0.475
0.450
0
10
20
30
40
50
60


Cleveland Cavaliers
Average Correct
Average Correct
Split used
0.60
0.58
0.56
0.54
0.52
0.50
0.48
0.46
0.44
0
10
20
30
40
50
60


Los Angeles Lakers
Average Correct
Average Correct
0.60
0.55
0.50
0.45


0.40
0
10
20
30
40
50
60
Split used
San Antonio Spurs

Fase IV. Modeling. Modelat

En aquesta fase, es seleccionen y s'utilitzen les tècniqeus de modelatge que siguin pertinents al problema (com més s'utilitzin millor), i es calibreen els seus paràmetres a valors òptims. Típicament hi ha diverses tècniques per el mateix problema de mineria de dades. Algunes tècniques tenen requeriments específics sobre la forma de les dades. Per tant casi sempre en qualsevol projecte s'acaba tornant a la fase de preparació de dades.

```
#Importem totes les llibreries requerides.

import pandas as pd
import pylab as pl
import numpy as np
import scipy.optimize as opt
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.metrics import jaccard_score
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import log_loss
import itertools
import seaborn as sns
import matplotlib as mpl
import matplotlib.pyplot as plt
%matplotlib inline
mpl.style.use('ggplot') # optional: for ggplot-like style
```

Obtenim les dades de la fase de preperació da dades que es troben el següent fitxer **.csv**.

```
from google.colab import files
uploaded = files.upload()
```

Elegir archivos Ningún archivo seleccionado Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving NBA_analysis_17_18.csv to NBA_analysis_17_18.csv

```
import io
dataframe = pd.read_csv(io.BytesIO(uploaded['NBA_analysis_17_18.csv']))

dataframe=dataframe[['Date','Visitor','PTSV','Home','PTSH','Overtime','Attend.','%W','%W as Home/Visitor','Back to back','Div_Visitor','Div_Home','Division','Cor

dataframe.head()
```

	Date	Visitor	PTSV	Home	PTSH	Overtime	Attend.	%W	%W as Home/Visitor	Back to back	Div_Visitor	Div_Home	Division	Conference_Visitor	Conference_Home	Conference	Last N	Net Rating	OT last match
0	2017-10-17	Boston Celtics	99	Cleveland Cavaliers	102	0	20,562	-0.024	0.102	0.0	1	2	1	1	1	0	-0.03	0.0	0.0
1	2017-10-17	Houston Rockets	122	Golden State Warriors	121	0	19,596	0.146	0.295	0.0	6	5	-1	0	0	0	0.15	0.0	0.0
2	2017-10-18	Charlotte Hornets	90	Detroit Pistons	102	0	20,491	0.012	0.101	0.0	3	2	-1	1	1	0	0.01	0.0	0.0
	2017-	Miami		Orlando															

A continuació creem un nou Dataframe per guardar els resultats dels tests que es faran a continuació.

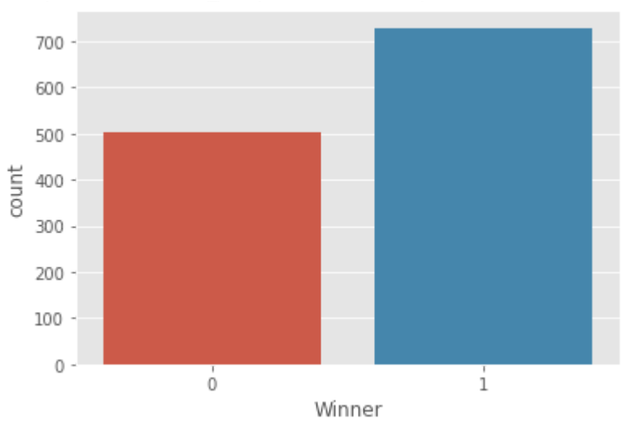
```
data = pd.DataFrame(columns=('Visitor','PTSV','Home','PTSH','New Date'))
```

Primer de tot volem realitzar un petit anàlisi visual a partir de les gràfiques per fer-nos una idea global de les dades que tenim.

En aquest primer gràfic analitzem com estan distribuides les victòries en funció de si el guanyador és l'equip local o el visitant. Ja que aquesta és la variable que voldem predir (Y). Recordem que el 0 siginfica victòria del visitant i el 1 victòria de l'equip local.

```
sns.countplot(x='Winner',data=dataframe)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f320ae44c88>
```

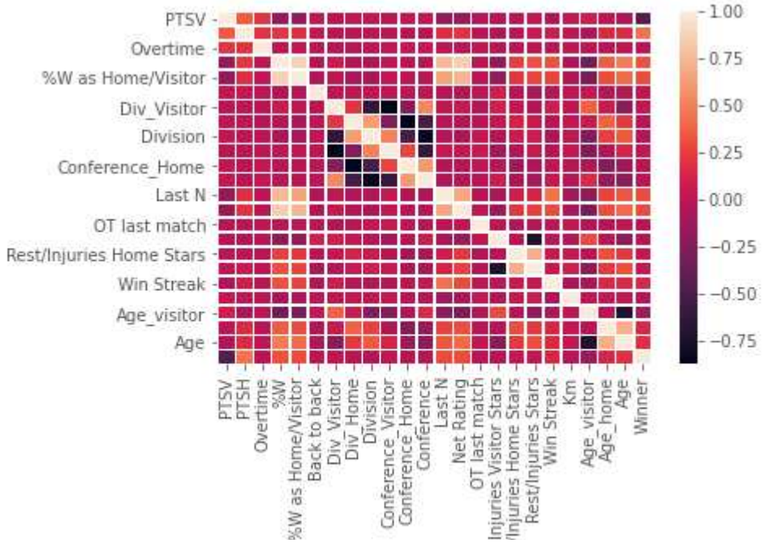


Podem observar com el nombre de victòries locals és significativament superior al nombre de victòries visitants. Per tant queda clar que el fet de jugar a casa és un factor que influeix en el resultat dels partits.

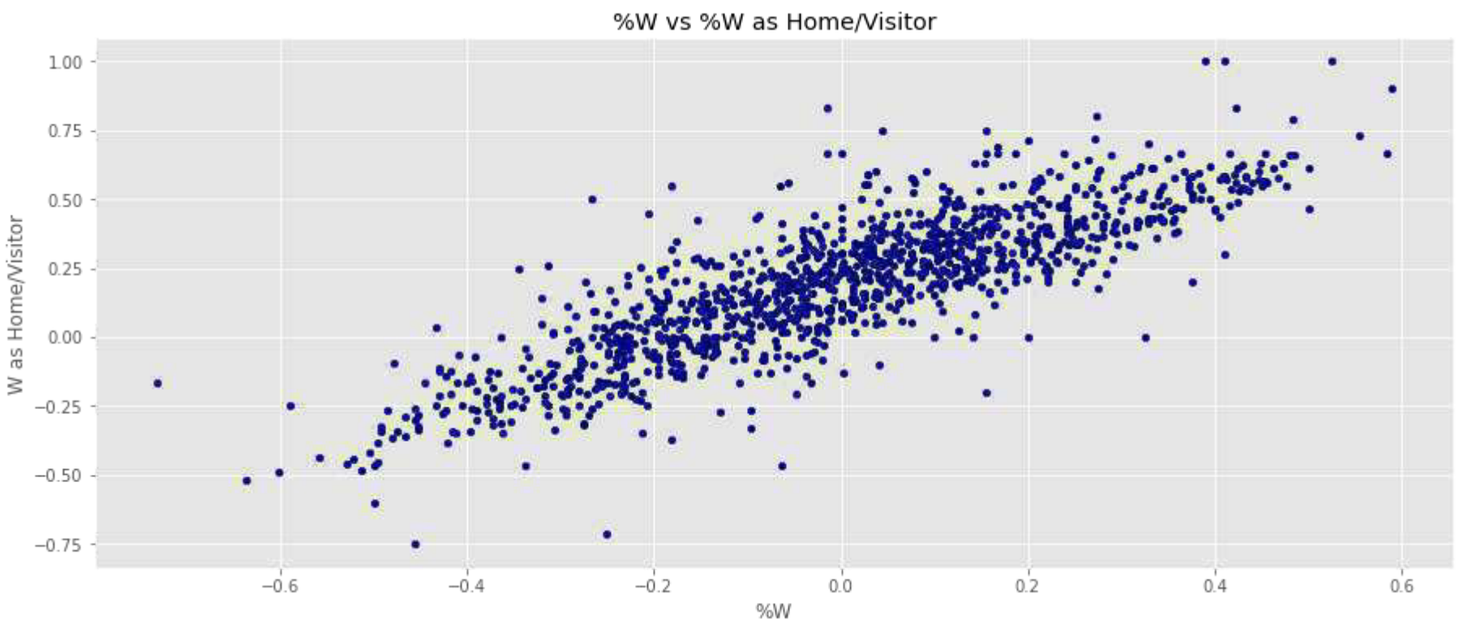
A continuació analitzem la correlació entre totes les variables per veure quines poden ser més rellevants a l'hora de introduir-les al model de predicció.

```
sns.heatmap(df, linewidths=.5)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f320b57eac8>
```

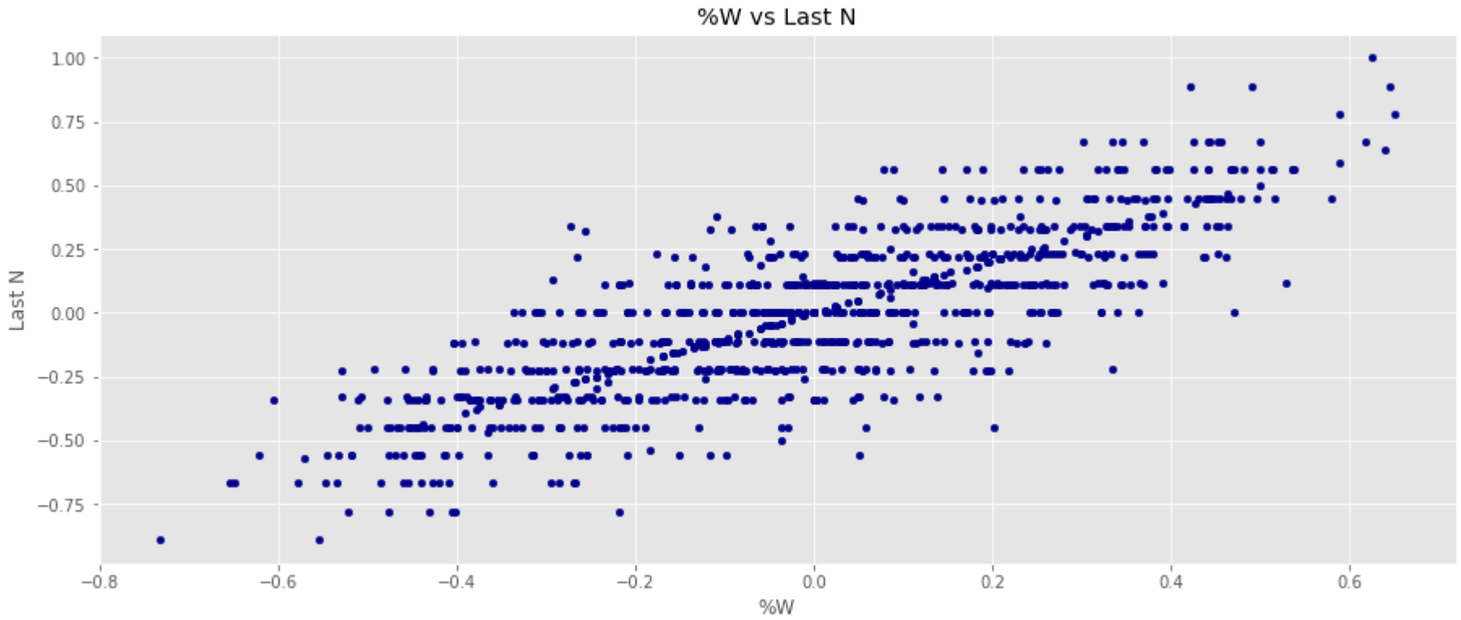



```
dataframe.plot(kind='scatter', x='%W', y='%W as Home/Visitor', figsize=(15,6), color='darkblue')

plt.title('%W vs %W as Home/Visitor')
plt.xlabel('%W')
plt.ylabel('W as Home/Visitor')

plt.show()
```

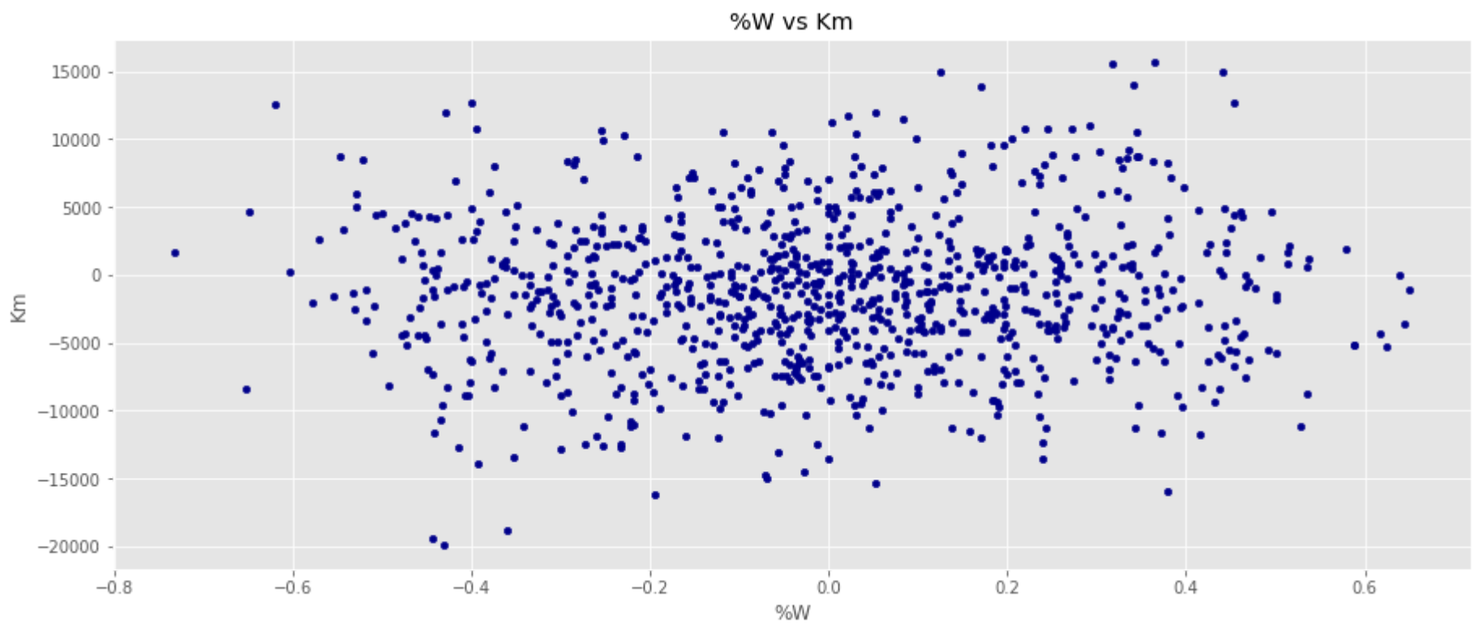



```
dataframe.plot(kind='scatter', x='%W', y='Last N', figsize=(15,6), color='darkblue')

plt.title('%W vs Last N')
plt.xlabel('%W')
plt.ylabel('Last N')
```

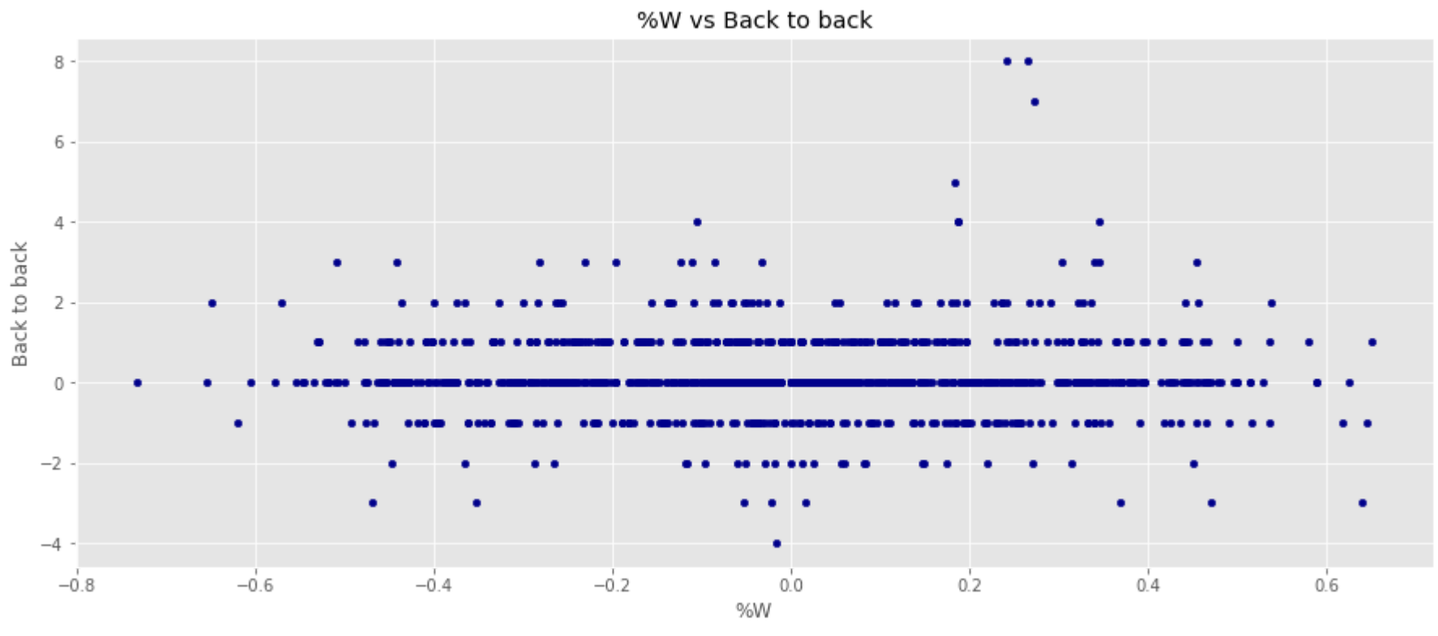
```
plt.show()
```




```
dataframe.plot(kind='scatter', x='%W', y='Km', figsize=(15,6), color='darkblue')

plt.title('%W vs Km')
plt.xlabel('%W')
plt.ylabel('Km')
plt.show()
```



```
dataframe.plot(kind='scatter', x='%W', y='Back to back', figsize=(15,6), color='darkblue')

plt.title('%W vs Back to back')
plt.xlabel('%W')
plt.ylabel('Back to back')
plt.show()
```



Quina variable ens dóna la millor predicció?

```
X=np.asarray(dataframe[['%W','%W as Home/Visitor','Back to back','Net Rating','OT last match',
                        'Rest/Injuries Stars','Last N','Win Streak','Km','Div_Visitor','Div_Home',
                        'Conference_Visitor','Conference_Home','Conference','Age_visitor','Age_home','Age']])
y = np.asarray(dataframe['Winner'])


X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.31, random_state=5)

print ('Train set:', X_train.shape,  y_train.shape)
print ('Test set:', X_test.shape,  y_test.shape)

LR = LogisticRegression(C=0.9, solver='liblinear').fit(X_train,y_train)
yhat = LR.predict(X_test)
yhat_prob = LR.predict_proba(X_test)

jac=jaccard_score(y_test, yhat)

selector = RFE(LR, 1, step=1)
selector = selector.fit(X, y)
support=selector.support_
```

```
ranking=selector.ranking_


features_to_select = pd.DataFrame(columns=('Features','Support','Ranking'))
features_to_select['Features']=np.array(['%W','%W as Home/Visitor','Back to back','Net Rating','OT last match',
                        'Rest/Injuries Stars','Last N','Win Streak','Km','Div_Visitor','Div_Home',
                        'Conference_Visitor','Conference_Home','Conference','Age_visitor','Age_home','Age'])
features_to_select['Support']=support
features_to_select['Ranking']=ranking
features=features_to_select.sort_values(by=['Ranking'])
features
```

```
Train set: (848, 17) (848,)
Test set: (382, 17) (382,)
```

	Features	Support	Ranking
0	%W	True	1
6	Last N	False	2
1	%W as Home/Visitor	False	3
5	Rest/Injuries Stars	False	4
11	Conference_Visitor	False	5
9	Div_Visitor	False	6
13	Conference	False	7
12	Conference_Home	False	8
10	Div_Home	False	9
16	Age	False	10
4	OT last match	False	11
2	Back to back	False	12
15	Age_home	False	13
14	Age_visitor	False	14
3	Net Rating	False	15
7	Win Streak	False	16
8	Km	False	17

Escollim les variables independents (X) que estan en un rang millor classificades en l'anàlisi RFE que hem realitzat anteriorment. També afegim la variable dependent (y) per el nostre model.

```
X = np.asarray(dataframe[['%W','%W as Home/Visitor','Last N','Rest/Injuries Stars']])
X[0:5]
```

```
array([[-0.02,  0.1 , -0.03,  1.  ],
       [ 0.15,  0.29,  0.15,  0.  ],
       [ 0.01,  0.1 ,  0.01,  0.  ],
```

```
       [ 0.27,  0.34,  0.27,  0.  ],
       [-0.15, -0.06, -0.15,  0.  ]])
```

```
y = np.asarray(dataframe['Winner'])
y [0:5]
```

```
array([1, 0, 1, 1, 1])
```

```
X = preprocessing.StandardScaler().fit(X).transform(X)
X[0:5]
```

```
array([[-0.11, -0.22, -0.11,  1.23],
       [ 0.68,  0.58,  0.47,  0.03],
       [ 0.06, -0.22,  0.02,  0.03],
       [ 1.24,  0.77,  0.85,  0.03],
       [-0.68, -0.89, -0.49,  0.03]])
```

Aquesta funció genera un plot de la confuson matrix (aquesta funció està extreta de un curset de machine learning no està programada per mi)

```
def plot_confusion_matrix(cm, classes,normalize=False,title='Confusion matrix',cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')
    print(cm)
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)
    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    print(confusion_matrix(y_test, yhat, labels=[1,0]))
    conf=(confusion_matrix(y_test, yhat, labels=[1,0]))
    conf=conf[0][0]+conf[1][1]
```

```
    print(conf)
    return conf
```

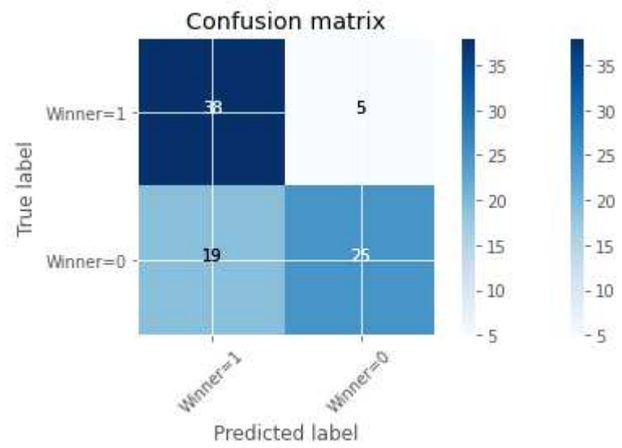
Anàlisi utilitzant l'algoritme Logistic Regression

Aquest loop repeteix la simulació N vegades, amb el propòsit per obtenir els resultats per una gran quantitat de números random, això ens dóna una estimació real de com el model està funcionant.

```
dev=[]
correct=[]
correct_average=[]
random=[]
N=200
adjustment=0
for m in range(N):
    X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.07, random_state=m+280)

    print ('Train set:', X_train.shape,  y_train.shape)
    print ('Test set:', X_test.shape,  y_test.shape)

    LR = LogisticRegression(C=0.9, solver='liblinear').fit(X_train,y_train)
    yhat = LR.predict(X_test)
    yhat_prob = LR.predict_proba(X_test)

    jaccard_score(y_test, yhat)

    # Compute confusion matrix
    cnf_matrix = confusion_matrix(y_test, yhat, labels=[1,0])
    np.set_printoptions(precision=2)

    # Plot non-normalized confusion matrix
    plt.figure()
    plot_confusion_matrix(cnf_matrix, classes=['Winner=1','Winner=0'],normalize= False,  title='Confusion matrix')
    ad=plot_confusion_matrix(cnf_matrix, classes=['Winner=1','Winner=0'],normalize= False,  title='Confusion matrix')
    adjustment=adjustment+ad
    random=np.append(random,m)
    correct=np.append(correct,ad)
    dev=np.append(dev,ad)
    print('Mitjana encerts')
    print(adjustment/(m+1))
    correct_average=np.append(correct_average,(adjustment/(m+1)))
    st_dev=np.std(dev)
    print('desviació estandar')
    print(st_dev)
    plt.show()

    print (classification_report(y_test, yhat))

    log_loss(y_test, yhat_prob)

    LR2 = LogisticRegression(C=0.01, solver='sag').fit(X_train,y_train)
```

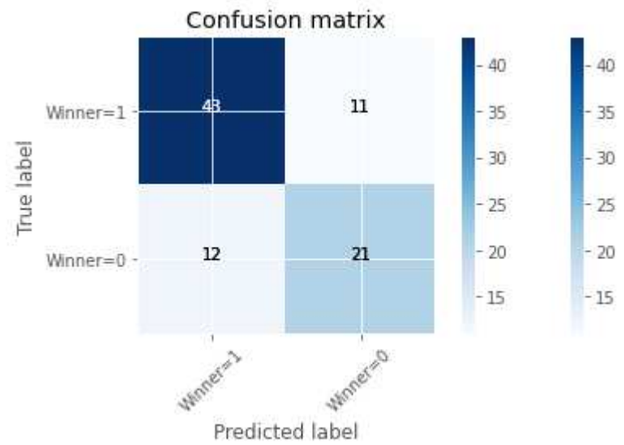
```
yhat_prob2 = LR2.predict_proba(X_test)
print ("LogLoss: : %.2f" % log_loss(y_test, yhat_prob2))
```

```
Train set: (1143, 4) (1143,)
Test set: (87, 4) (87,)
Confusion matrix, without normalization
[[38  5]
 [19 25]]
[[38  5]
 [19 25]]
63
Confusion matrix, without normalization
[[38  5]
 [19 25]]
[[38  5]
 [19 25]]
63
Mitjana encerts
63.0
desviació estandar
0.0
```

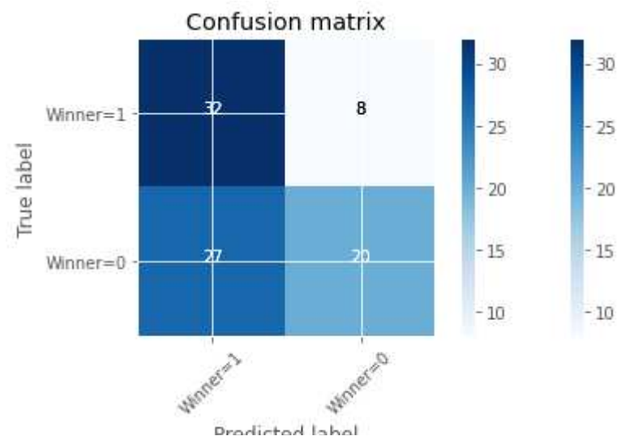



```
              precision    recall  f1-score   support

           0       0.83      0.57      0.68        44
           1       0.67      0.88      0.76        43

    accuracy                           0.72        87
   macro avg       0.75      0.73      0.72        87
weighted avg       0.75      0.72      0.72        87

LogLoss: : 0.61
Train set: (1143, 4) (1143,)
Test set: (87, 4) (87,)
Confusion matrix, without normalization
[[43 11]
 [12 21]]
[[43 11]
 [12 21]]
64
Confusion matrix, without normalization
[[43 11]
 [12 21]]
[[43 11]
 [12 21]]
64
Mitjana encerts
63.5
desviació estandar
```

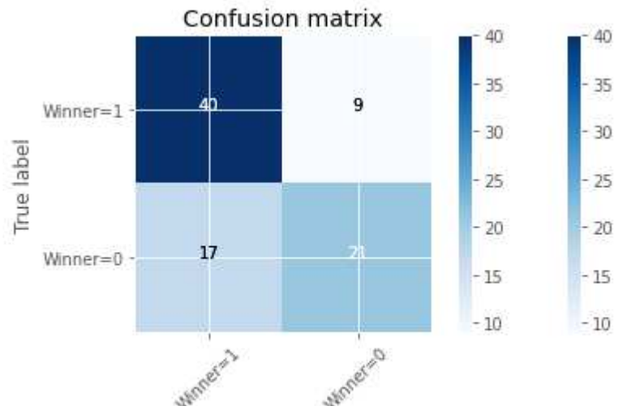
```
desviació estandar
0.5
```




```
              precision    recall  f1-score   support

           0       0.66      0.64      0.65        33
           1       0.78      0.80      0.79        54

    accuracy                           0.74        87
   macro avg       0.72      0.72      0.72        87
weighted avg       0.73      0.74      0.73        87

LogLoss: : 0.58
Train set: (1143, 4) (1143,)
Test set: (87, 4) (87,)
Confusion matrix, without normalization
[[32  8]
 [27 20]]
[[32  8]
 [27 20]]
52
Confusion matrix, without normalization
[[32  8]
 [27 20]]
[[32  8]
 [27 20]]
52
Mitjana encerts
59.666666666666664
desviació estandar
5.436502143433364
```



```
           1       0.53      0.67      0.59        45

    accuracy                           0.52        87
   macro avg       0.51      0.51      0.50        87
weighted avg       0.51      0.52      0.51        87

LogLoss: : 0.68
Train set: (1143, 4) (1143,)
Test set: (87, 4) (87,)
Confusion matrix, without normalization
[[40  9]
 [17 21]]
[[40  9]
 [17 21]]
61
Confusion matrix, without normalization
[[40  9]
 [17 21]]
[[40  9]
 [17 21]]
61
Mitjana encerts
55.93467336683417
desviació estandar
4.294527751907506
```

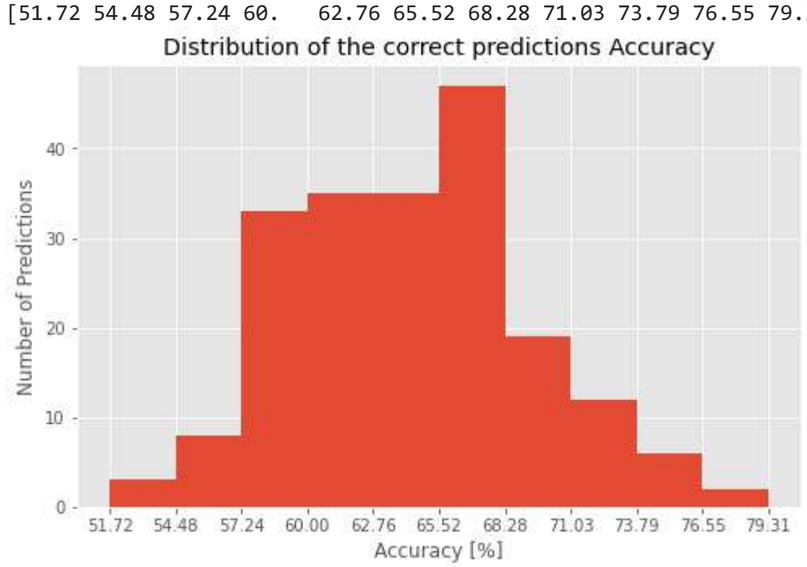



```
# let's quickly view the data
dframe=pd.DataFrame(columns=['Random State','Accuracy (%)'])
dframe['Random State']=random
dframe['Accuracy (%)']=correct/float(X_test.shape[0])*100
dframe['Correct']=correct

dframe
```

	Random State	Accuracy (%)	Correct
0	0.0	72.413793	63.0
1	1.0	73.563218	64.0
2	2.0	59.770115	52.0
3	3.0	62.068966	54.0
4	4.0	60.919540	53.0
...	...	...	...

```
# 'bin_edges' is a list of bin intervals
count, bin_edges = np.histogram(dframe['Accuracy (%)'],bins=int(10))
dframe['Accuracy (%)'].plot(kind='hist', figsize=(8, 5),xticks=bin_edges)

print(count) # frequency count
print(bin_edges) # bin ranges=test.shape/2

plt.title('Distribution of the correct predictions Accuracy') # add a title to the histogram
plt.ylabel('Number of Predictions') # add y-label
plt.xlabel('Accuracy [%]') # add x-label

plt.show()
```

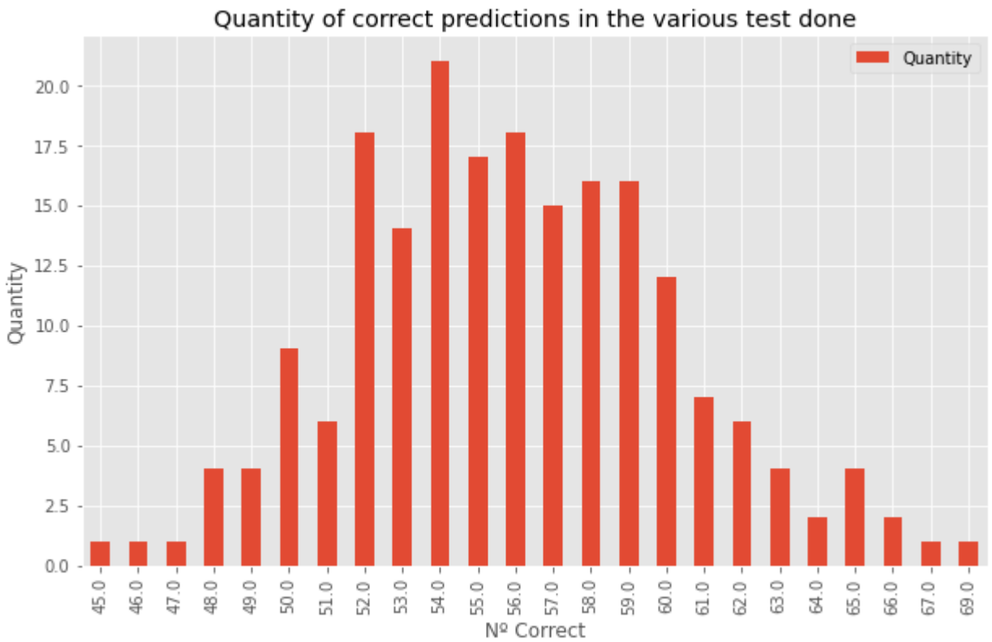
```
[ 3  8 33 35 35 47 19 12  6  2]
[51.72 54.48 57.24 60.   62.76 65.52 68.28 71.03 73.79 76.55 79.31]
```




```
df_corr=dframe.groupby('Correct').count()
df_corr.drop(['Accuracy (%)'],axis=1,inplace=True)
df_corr.rename(columns={"Random State": "Quantity"},inplace=True)
df_corr['Quantity']
print(type(df_corr))
```

```
<class 'pandas.core.frame.DataFrame'>
```

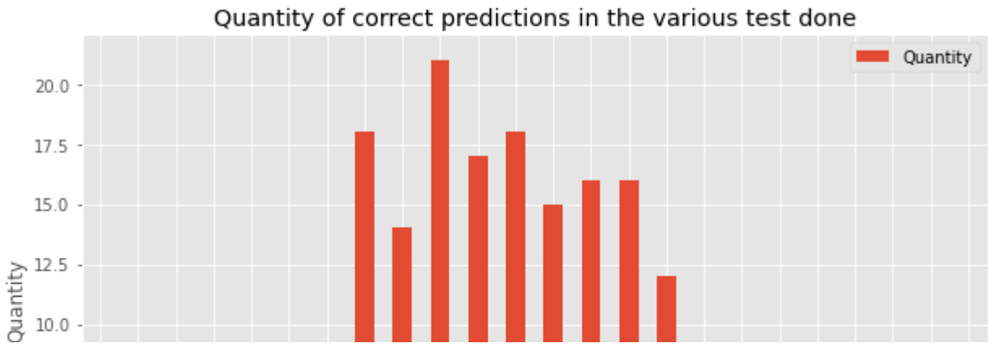
```
#print(df_corr.loc['Random State', 'Nº Correct'])
# step 2: plot data
df_corr.plot(kind='bar', figsize=(10, 6))

plt.xlabel('Nº Correct') # add to x-label to the plot
plt.ylabel('Quantity') # add y-label to the plot
plt.title('Quantity of correct predictions in the various test done') # add title to the plot

plt.show()
```




```
df_corr
```

Correct	Quantity
45.0	1
46.0	1
47.0	1
48.0	4
49.0	4
50.0	9
51.0	6
52.0	18
53.0	14
54.0	21
55.0	17
56.0	18
57.0	15

```
#print(df_corr.loc['Random State', 'Nº Correct'])
# step 2: plot data
colors=[]
for i,e in enumerate(df_corr['Quantity']):
    if i<=8:
        colors=colors+['red']
    else:
        colors=colors+['green']

df_corr.plot(kind='bar', figsize=(10, 6))

plt.xlabel('Nº Correct') # add to x-label to the plot
plt.ylabel('Quantity') # add y-label to the plot
plt.title('Quantity of correct predictions in the various test done') # add title to the plot



plt.show()
```



How many test we have to do to get a correct accuracy?

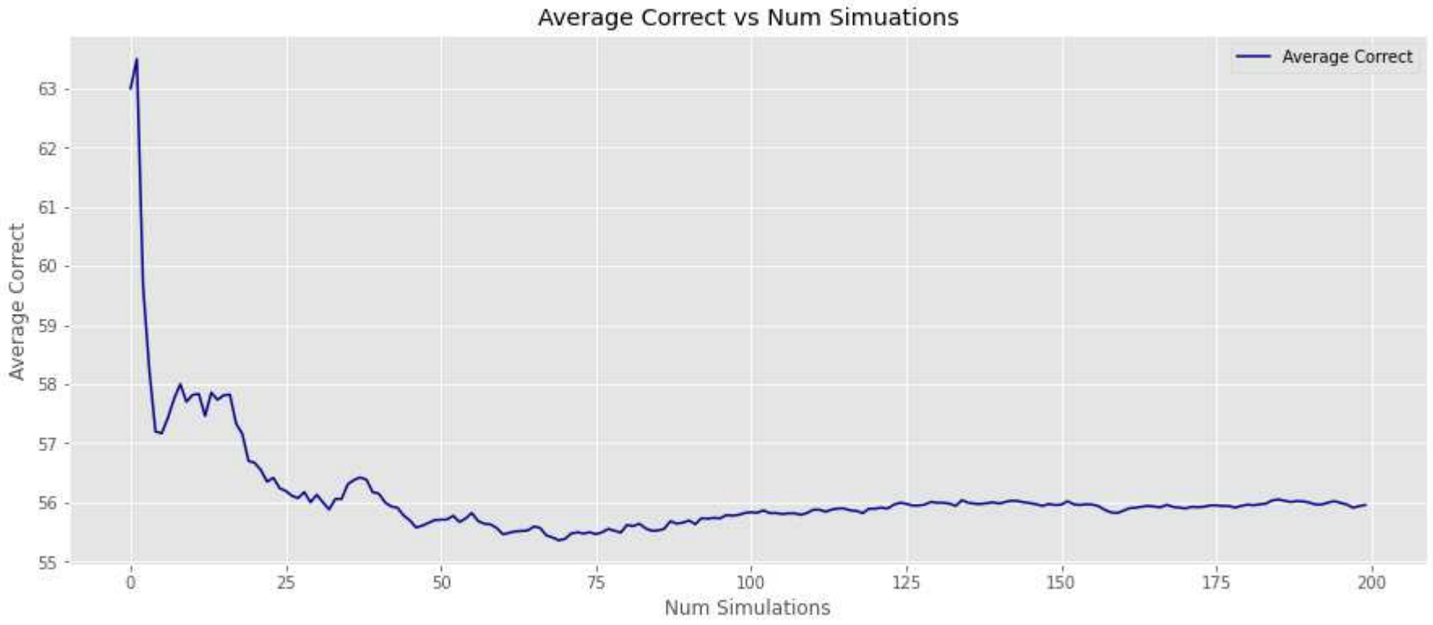
```
average=pd.DataFrame()
average['Average Correct']=correct_average
average
```

	Average Correct
0	63.000000
1	63.500000
2	59.666667
3	58.250000
4	57.200000
...	...
195	55.989796
196	55.964467
197	55.909091
198	55.934673
199	55.955000

200 rows × 1 columns

```
average.plot(kind='line', y='Average Correct', figsize=(15,6), color='darkblue')

plt.title('Average Correct vs Num Simuations')
plt.xlabel('Num Simulations')
plt.ylabel('Average Correct')

# Annotate Text
plt.annotate('Stable average Num iter = 50', # text to display
             xy=(55, 42.9),                    # start the text at at point (year 2008 , pop 30)
             rotation=0,                  # based on trial and error to match the arrow
             va='bottom',                    # want the text to be vertically 'bottom' aligned
             ha='left',                      # want the text to be horizontally 'left' algned.
            )
```

```
plt.show()
```


Average Correct vs Num Simuations
Average Correct
Average Correct
Num Simulations
63
62
61
60
59
58
57
56
55
0
25
50
75
100
125
150
175
200