In [56]:
```python
import numpy as np
import pandas as pd
from pandas import read_csv
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import r2_score
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.ensemble import RandomForestRegressor
import seaborn as sns
import math
from sklearn.cluster import KMeans
import statistics as stat
from sklearn.neural_network import MLPRegressor
from sklearn.decomposition import PCA
```

Seguimos el hilo argumental del ejercicio 9.1 y ejercicio 9.2 para trabajarcon el ejercicio 10 . Primero cargamos el Data Set.

In [3]:
```python
df = pd.read_csv("DelayedFlights.csv") # este es el conjunto de datos proporcionado en el ejercicio
df.head(10)
```

Out[3]:

	Unnamed: 0	Year	Month	DayofMonth	DayOfWeek	DepTime	CRSDepTime	ArrTime	CRSArrTime	UniqueCarrier	...	TaxiIn	TaxiOut
0	0	2008	1	3	4	2003.0	1955	2211.0	2225	WN	...	4.0	8.0
1	1	2008	1	3	4	754.0	735	1002.0	1000	WN	...	5.0	10.0
2	2	2008	1	3	4	628.0	620	804.0	750	WN	...	3.0	17.0
3	4	2008	1	3	4	1829.0	1755	1959.0	1925	WN	...	3.0	10.0
4	5	2008	1	3	4	1940.0	1915	2121.0	2110	WN	...	4.0	10.0
5	6	2008	1	3	4	1937.0	1830	2037.0	1940	WN	...	3.0	7.0
6	10	2008	1	3	4	706.0	700	916.0	915	WN	...	5.0	19.0
7	11	2008	1	3	4	1644.0	1510	1845.0	1725	WN	...	6.0	8.0
8	15	2008	1	3	4	1029.0	1020	1021.0	1010	WN	...	6.0	9.0
9	16	2008	1	3	4	1452.0	1425	1640.0	1625	WN	...	7.0	8.0

10 rows × 30 columns

1. **Tratamiento de variables**.

En este apartado vamos a hacer tratamiento de variables para luego aplicar a programas de Clustering

Hemos dejado los enlaces del ejercicio 9.1 y ejercicio 9.2 en que se explica los razonamientos para sleccionar una variable un otra,

aunque volveremos a exolicar los motivos por los que se seleccionan variables, unas sí u otras no, no entraremos tan en detalle

a) Variable Unnamed 0 y Year, básicamente son un índice y el año de vuelos del 2008. Year es una constante. Así que las eliminamos.

b) las variables "UniqueCarrier', 'FlightNum','TailNum, 'Month', 'DayofMonth', 'DayOfWeek', 'Cancelled', 'CancellationCode', "Diverted", "Origin", "Dest" *son eliminadas por siguiente motivo. KMeans funciona calculando distancias en un espacio ndimensional, de volumen R^n. Si implementamos variables categóricas, por ejemplo binarias, o con más clases como los meses, los puntos van a estar concentrados en hiperplanos de dimensión n- 1. Si las variables no categóricas, están normalizadas o estandarizadas, la distancia entre hiperplanos será por lo general mayor , que la distancia entre puntos dentro de un hiperplano de dimensión n-1.*

c) También eliminamos 'ActualElapsedTime', 'CRSElapsedTime', 'AirTime' ya que tienen un 0.95 de correlación o más con Distance y aportan la misma información. ActualElapsed es el tiempo esperado total del vuelo(desembarco, salida, vuelo, más atterizaje), CRSElapsedTime es el mismo tiempo previsto, mientras que AirTime es el tiempo que el avión está en el aire y Distance la distancia recorrida en millas.

In [4]:
```python
df[["ActualElapsedTime", 'CRSElapsedTime', 'AirTime' , "Distance" ]].corr()
```

Out[4]:

	ActualElapsedTime	CRSElapsedTime	AirTime	Distance
ActualElapsedTime	1.000000	0.971122	0.976660	0.952980
CRSElapsedTime	0.971122	1.000000	0.986086	0.981759
AirTime	0.976660	0.986086	1.000000	0.980294
Distance	0.952980	0.981759	0.980294	1.000000

In [5]:
```python
df1= df.drop(["Unnamed: 0", "Year", "UniqueCarrier", 'FlightNum',"TailNum", "Month", "DayofMonth",
             'DayOfWeek', 'Cancelled', 'CancellationCode', "Diverted", "Origin", "Dest","ActualElapsedTime", '
df1.head(10)
```

Out[5]:

	DepTime	CRSDepTime	ArrTime	CRSArrTime	ArrDelay	DepDelay	Distance	TaxiIn	TaxiOut	CarrierDelay	WeatherDelay	NASDelay
0	2003.0	1955	2211.0	2225	-14.0	8.0	810	4.0	8.0	NaN	NaN	NaN
1	754.0	735	1002.0	1000	2.0	19.0	810	5.0	10.0	NaN	NaN	NaN
2	628.0	620	804.0	750	14.0	8.0	515	3.0	17.0	NaN	NaN	NaN
3	1829.0	1755	1959.0	1925	34.0	34.0	515	3.0	10.0	2.0	0.0	0.0
4	1940.0	1915	2121.0	2110	11.0	25.0	688	4.0	10.0	NaN	NaN	NaN
5	1937.0	1830	2037.0	1940	57.0	67.0	1591	3.0	7.0	10.0	0.0	0.0
6	706.0	700	916.0	915	1.0	6.0	828	5.0	19.0	NaN	NaN	NaN
7	1644.0	1510	1845.0	1725	80.0	94.0	828	6.0	8.0	8.0	0.0	0.0
8	1029.0	1020	1021.0	1010	11.0	9.0	162	6.0	9.0	NaN	NaN	NaN
9	1452.0	1425	1640.0	1625	15.0	27.0	1489	7.0	8.0	3.0	0.0	0.0

Como vamos a empezar a transformar variables, vamos a hacer previamente un muestreo.

In [6]:
```python
df2= df1.sample (390000,random_state=55)
```

Cómo vimos en los ejercicios 9.1 y 9.2, los valores NaN de Arrdelay, así como de otras variables coincidían con aquellos que el vuelo había desviado o cancelado, valores imposibles de deducir (los NaN) por el mismo concepto de cancelación o desvió. por lo que eliminamos los valores NaN de ArrDelay

In [7]:
```python
df3=df2.dropna( subset=["ArrDelay"]).reset_index(drop=True)
df3.isna().sum()
```

Out[7]:
```
DepTime                   0
CRSDepTime                0
ArrTime                   0
CRSArrTime                0
ArrDelay                  0
DepDelay                  0
Distance                  0
TaxiIn                    0
TaxiOut                   0
CarrierDelay         137592
WeatherDelay         137592
NASDelay             137592
SecurityDelay        137592
LateAircraftDelay    137592
dtype: int64
```

In [8]:
```python
df3.shape
```

Out[8]:
```
(388357, 14)
```

0.1. **Transformación de variables horarias**

En esta parte vamos a convertir las variables DepTime, CRSDepTime, ArrTime, CRSArrTime en la función cíclica.

Estas cuatro variables vienen en formato horario hh:mm. Lo que haremos será contar todos los minutos transcurridos durante

el día, siendo 0 minutos a las 00:00 y 1440 los minutos transcurridos durante el día a las 23:59.

Más adelante, en el apartado 0.4, transformaremos estas variables en cíclicas.

In [9]:
```python
# Primero de todo convierto las variables horarias en formato hora y para eso tienen que haber 4 digitos, que 
#izquierda con ceros
# Primero tengo que convertir en entero las variables DepTime y ArrTime en enteros para evitar los decimales


df3['DepTime'] = df3['DepTime'].astype(int)
```

In [10]:
```python
df3['ArrTime'] = df3['ArrTime'].astype(int)
```

In [11]:
```python
#relleno por la izquierda con ceros
df3['DepTime'] = df3['DepTime'].astype(str).str.zfill(4)
df3['CRSDepTime'] = df3['CRSDepTime'].astype(str).str.zfill(4)
df3['ArrTime'] = df3['ArrTime'].astype(str).str .zfill(4)
df3['CRSArrTime'] = df3['CRSArrTime'].astype(str).str.zfill(4)
df3.head()
```

Out[11]:

	DepTime	CRSDepTime	ArrTime	CRSArrTime	ArrDelay	DepDelay	Distance	TaxiIn	TaxiOut	CarrierDelay	WeatherDelay	NASDelay
0	0918	0905	1044	1035	9.0	13.0	414	3.0	12.0	NaN	NaN	NaN
1	1421	1400	1500	1450	10.0	21.0	192	1.0	7.0	NaN	NaN	NaN
2	2250	2153	0013	2330	43.0	57.0	432	5.0	17.0	0.0	43.0	0.0
3	0049	0040	0521	0530	-9.0	9.0	1235	4.0	17.0	NaN	NaN	NaN
4	1854	1845	2017	2010	7.0	9.0	488	4.0	11.0	NaN	NaN	NaN

In [12]:
```python
# las convierto en formato horario( Nota: en un principio lo pasé a formato horario por si lo necesitaba para 
# al final opté por otro tipo de conversión)
df3['DepTime'] = df3['DepTime'].astype(str).str[:2]  + ':' + df3['DepTime'].astype(str).str[2:4] + ':00'
df3['CRSDepTime'] = df3['CRSDepTime'].astype(str).str[:2] + ':' + df3['CRSDepTime'].astype(str).str[2:4] + ':0
df3['ArrTime'] = df3['ArrTime'].astype(str).str[:2] + ':' + df3['ArrTime'].astype(str).str[2:4]  + ':00'
df3['CRSArrTime'] = df3['CRSArrTime'].astype(str).str[:2] + ':' + df3['CRSArrTime'].astype(str).str[2:4] + ':0


df3
```

Out[12]:

	DepTime	CRSDepTime	ArrTime	CRSArrTime	ArrDelay	DepDelay	Distance	TaxiIn	TaxiOut	CarrierDelay	WeatherDelay	NASDelay
0	09:18:00	09:05:00	10:44:00	10:35:00	9.0	13.0	414	3.0	12.0	NaN	NaN	NaN
1	14:21:00	14:00:00	15:00:00	14:50:00	10.0	21.0	192	1.0	7.0	NaN	NaN	NaN
2	22:50:00	21:53:00	00:13:00	23:30:00	43.0	57.0	432	5.0	17.0	0.0	43.0	0.0
3	00:49:00	00:40:00	05:21:00	05:30:00	-9.0	9.0	1235	4.0	17.0	NaN	NaN	NaN
4	18:54:00	18:45:00	20:17:00	20:10:00	7.0	9.0	488	4.0	11.0	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...
388352	08:59:00	08:45:00	11:01:00	10:20:00	41.0	14.0	1333	5.0	15.0	14.0	0.0	27.0
388353	14:59:00	14:50:00	16:52:00	16:50:00	2.0	9.0	594	2.0	14.0	NaN	NaN	NaN
388354	21:32:00	20:30:00	22:43:00	21:30:00	73.0	62.0	255	5.0	14.0	11.0	0.0	11.0
388355	19:42:00	17:50:00	21:18:00	19:35:00	103.0	112.0	583	3.0	14.0	83.0	0.0	0.0
388356	08:15:00	07:45:00	09:32:00	09:05:00	27.0	30.0	427	5.0	13.0	0.0	0.0	27.0

388357 rows × 14 columns

In [13]:
```python
# creamos la función minutos, que divide la hora hh:mm con un Split, en una lista ("hh","mm"), reconvierte hh 
# para luego pasarlos a minutos, y con la reconverión ya comentada aplica la función minutos()
def minutos(x):
    x=x.split( sep=":")
    seg= 60*(int(x[0]))+(int(x[1]))

    return seg



dfhoras= df3[["DepTime", "CRSDepTime", "ArrTime", "CRSArrTime"]]
```

In [14]:
```python
dfhoras_DT= dfhoras["DepTime"].apply(minutos)
dfhoras_CRSD=dfhoras["CRSDepTime"].apply(minutos)
dfhoras_AT=dfhoras["ArrTime"].apply(minutos)
dfhoras_CRSA=dfhoras["CRSArrTime"].apply(minutos)
```

In [15]:
```python
df4= df3.drop([ 'DepTime',
       'CRSDepTime', 'ArrTime', 'CRSArrTime'], axis=1)
```

In [16]:
```python
# ahora añadimos las cuatro columnas nuevas

df5= pd.concat([df4, dfhoras_DT,dfhoras_CRSD , dfhoras_AT, dfhoras_CRSA], axis=1)
df5.columns
```

Out[16]:
```
Index(['ArrDelay', 'DepDelay', 'Distance', 'TaxiIn', 'TaxiOut', 'CarrierDelay',
       'WeatherDelay', 'NASDelay', 'SecurityDelay', 'LateAircraftDelay',
       'DepTime', 'CRSDepTime', 'ArrTime', 'CRSArrTime'],
      dtype='object')
```

In [17]:
```python
df5[["DepTime", "CRSDepTime", "ArrTime", "CRSArrTime"]].describe()# miramos como quedan para ver si hay alguna
# los máximos y mínimos
```

Out[17]:

	DepTime	CRSDepTime	ArrTime	CRSArrTime
count	388357.000000	388357.00000	388357.000000	388357.000000
mean	922.620877	891.35868	978.678528	992.171942
std	269.858220	255.01740	327.865454	278.288941
min	1.000000	0.00000	1.000000	0.000000
25%	723.000000	695.00000	797.000000	805.000000
50%	944.000000	910.00000	1035.000000	1025.000000
75%	1140.000000	1095.00000	1230.000000	1213.000000
max	1440.000000	1439.00000	1440.000000	1439.000000

In [18]:
```python
df5.isna().sum()
```

Out[18]:
```
ArrDelay                  0
DepDelay                  0
Distance                  0
TaxiIn                    0
TaxiOut                   0
CarrierDelay         137592
WeatherDelay         137592
NASDelay             137592
SecurityDelay        137592
LateAircraftDelay    137592
DepTime                   0
CRSDepTime                0
ArrTime                   0
CRSArrTime                0
dtype: int64
```

0.2 **Valores NaN**

En esta sección vamos a completar las variables del motivo del retraso que tiene varios NaN. Como ya pudimos observar en el ejercicio 9.2, los vuelos con retrasos de menos de 14 minutos, tienen valores Nan

Carrier Delay es el retraso de la compañía

WeatherDelay es el retraso por las condiciones climatológicas SecurityDelay es el retraso por cuestiones de seguridad

LateAircraftDelay es el retraso de la misma aeronave.

Nas delay son los retraso causado por el Sistema Nacional del Espacio Aéreo (NAS)

por lo que vamos a asigarn 0 a los valores Nan

In [19]:
```python
df_delay= df5[['ArrDelay','DepDelay','CarrierDelay', 'WeatherDelay', 'NASDelay', 'SecurityDelay','LateAircraftD
df5_0= df5.drop(  ["CarrierDelay", 'WeatherDelay', 'NASDelay', 'SecurityDelay','LateAircraftDelay'], axis=1 )
delay_not_NAN= df_delay[['CarrierDelay', 'WeatherDelay', 'NASDelay', 'SecurityDelay','LateAircraftDelay']].fill
```

In [20]:
```python
df5= pd.concat([df5_0, delay_not_NAN], axis =1)
df5.head(10)
```

Out[20]:

	ArrDelay	DepDelay	Distance	TaxiIn	TaxiOut	DepTime	CRSDepTime	ArrTime	CRSArrTime	CarrierDelay	WeatherDelay	NASDelay
0	9.0	13.0	414	3.0	12.0	558	545	644	635	0.0	0.0	0.0
1	10.0	21.0	192	1.0	7.0	861	840	900	890	0.0	0.0	0.0
2	43.0	57.0	432	5.0	17.0	1370	1313	13	1410	0.0	43.0	0.0
3	-9.0	9.0	1235	4.0	17.0	49	40	321	330	0.0	0.0	0.0
4	7.0	9.0	488	4.0	11.0	1134	1125	1217	1210	0.0	0.0	0.0
5	134.0	105.0	316	15.0	39.0	1150	1045	1219	1085	0.0	0.0	87.0
6	-15.0	8.0	1235	5.0	16.0	768	760	997	1012	0.0	0.0	0.0
7	32.0	40.0	304	5.0	13.0	440	400	502	470	32.0	0.0	0.0
8	24.0	17.0	1589	5.0	29.0	527	510	654	630	17.0	0.0	7.0
9	49.0	36.0	787	7.0	16.0	1226	1190	1369	1320	6.0	0.0	13.0

In [21]:
```python
df5.isna().sum()
```

Out[21]:
```
ArrDelay             0
DepDelay             0
Distance             0
TaxiIn               0
TaxiOut              0
DepTime              0
CRSDepTime           0
ArrTime              0
CRSArrTime           0
CarrierDelay         0
WeatherDelay         0
NASDelay             0
SecurityDelay        0
LateAircraftDelay    0
dtype: int64
```

0.3 **dos nuevas variables.**

Se puede ver a simple vista que ArrDelay y DepDelay se obtienen de la resta entre (DepTime-CRSDepTime)y (ArrTime-CRSArrTime), así que calculamos dos nuevas variables, para reducir la dimensionalidad.

Primero miramos el valor más bajo de ArrDelay, para poder hacer los cálculo correctamente

In [22]:
```python
minimo=df[df["ArrDelay"]<0].sort_values("ArrDelay")# miramos los valores más bajos de ArrDelay
zmin=minimo["ArrDelay"].min()
zmin
```

Out[22]:
```
-109.0
```

In [23]:
```python
def rest(z):
    x=z[0]
    y=z[1]
    if (x < y) & ((x-y)<zmin):
        t= (1440+x)-y
        return t
    else:
        t= x-y
        return t

x11= df5[["DepTime","CRSDepTime" ]].apply(rest, axis=1)
x10= df5[["ArrTime","CRSArrTime" ]].apply(rest,axis=1)

x10=x10.rename("X10")
x11=x11.rename("X11")
x10.describe()
```

Out[23]:
```
count    388357.000000
mean         42.414732
std          60.245467
min         -96.000000
25%           9.000000
50%          24.000000
75%          56.000000
max        1439.000000
Name: X10, dtype: float64
```

In [24]:
```python
x11.describe()
```

Out[24]:
```
count    388357.000000
mean         42.990375
std          52.804525
min         -82.000000
25%          12.000000
50%          24.000000
75%          53.000000
max        1280.000000
Name: X11, dtype: float64
```

Sean **x10** y **x11** definidas por

x10= df6["ArrTime"]-df6["CRSArrTime"]

x11= df6["DepTime"]-df6["CRSDepTime"] y tienen una relación lineal con ArrDelay y DepDeplay respectivamente

In [25]:
```python
x1= pd.concat([x10,x11,df5["ArrDelay"],df5["DepDelay"] ],axis=1)
x1.corr()
```

Out[25]:

	X10	X11	ArrDelay	DepDelay
X10	1.000000	0.885914	0.925995	0.884185
X11	0.885914	1.000000	0.951128	0.998096
ArrDelay	0.925995	0.951128	1.000000	0.952835
DepDelay	0.884185	0.998096	0.952835	1.000000

In [26]:
```python
# remodelamos el data set con las dos nuevas variables
df5b=df5.drop(['DepTime', 'CRSDepTime', 'ArrTime', 'CRSArrTime'], axis=1)
df6=pd.concat([df5b, x10,x11], axis =1)
df6.head(10)
```

Out[26]:

	ArrDelay	DepDelay	Distance	TaxiIn	TaxiOut	CarrierDelay	WeatherDelay	NASDelay	SecurityDelay	LateAircraftDelay	X10	X11
0	9.0	13.0	414	3.0	12.0	0.0	0.0	0.0	0.0	0.0	9	13
1	10.0	21.0	192	1.0	7.0	0.0	0.0	0.0	0.0	0.0	10	21
2	43.0	57.0	432	5.0	17.0	0.0	43.0	0.0	0.0	0.0	43	57
3	-9.0	9.0	1235	4.0	17.0	0.0	0.0	0.0	0.0	0.0	-9	9
4	7.0	9.0	488	4.0	11.0	0.0	0.0	0.0	0.0	0.0	7	9
5	134.0	105.0	316	15.0	39.0	0.0	0.0	87.0	0.0	47.0	134	105
6	-15.0	8.0	1235	5.0	16.0	0.0	0.0	0.0	0.0	0.0	-15	8
7	32.0	40.0	304	5.0	13.0	32.0	0.0	0.0	0.0	0.0	32	40
8	24.0	17.0	1589	5.0	29.0	17.0	0.0	7.0	0.0	0.0	24	17
9	49.0	36.0	787	7.0	16.0	6.0	0.0	13.0	0.0	30.0	49	36

In [27]:
```python
df6.shape
```

Out[27]:
```
(388357, 12)
```

En el siguiente paso vamos a analizar la multicolinealidad. Para ver si alguna variable tiene un multicolinealidad muy elevada VIFi = 1/ (1 - Ri^2) donde Ri es el coeficiente de determinación de la regresión lineal

In [38]:
```python
from statsmodels.stats.outliers_influence import variance_inflation_factor

def vif(X):
    vifDF = pd.DataFrame()
    vifDF["variables"] = X.columns
    vifDF["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
    return vifDF

round(vif(df6),2)
```

Out[38]:

	variables	VIF
0	ArrDelay	230.80
1	DepDelay	478.27
2	Distance	2.71
3	TaxiIn	2.24
4	TaxiOut	3.30
5	CarrierDelay	59.28
6	WeatherDelay	13.08
7	NASDelay	35.59
8	SecurityDelay	1.11
9	LateAircraftDelay	62.56
10	X10	10.75
11	X11	449.33

0.4 **Reducción de la dimensionalidad por PCA**

Vamos a reducir las dimensiones para realizar el Cluster y bajar las 12 dimensiones lo máximo posible sin perder Varianza. Primero de todo vamos a estandarizar los datos para que no haya una componente que domine sobre el resto. Luego miraremos la perdida de Varianza Explicada

In [116…
```python
scaler = StandardScaler()
scaler.fit(df6)
df7=pd.DataFrame(scaler.transform(df6), columns=df6.columns)
df7.head()
```

Out[116…

	ArrDelay	DepDelay	Distance	TaxiIn	TaxiOut	CarrierDelay	WeatherDelay	NASDelay	SecurityDelay	LateAircraftDelay	X10
0	-0.585571	-0.566363	-0.612104	-0.725974	-0.434897	-0.343256	-0.138081	-0.346468	-0.036093	-0.456058	-0.554644
1	-0.567867	-0.415322	-0.999535	-1.108255	-0.785606	-0.343256	-0.138081	-0.346468	-0.036093	-0.456058	-0.538045
2	0.016337	0.264364	-0.580690	-0.343693	-0.084187	-0.343256	2.358603	-0.346468	-0.036093	-0.456058	0.009715
3	-0.904228	-0.641884	0.820694	-0.534833	-0.084187	-0.343256	-0.138081	-0.346468	-0.036093	-0.456058	-0.853422
4	-0.620977	-0.641884	-0.482960	-0.534833	-0.505039	-0.343256	-0.138081	-0.346468	-0.036093	-0.456058	-0.587841

In [117…
```python
pca =PCA().fit(df7)
varexp= pca.explained_variance_ratio_.cumsum()
plt.plot(varexp)
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance');
```

por lo visto, escogeremos 6 compnentes principales, en que la ganancia de Varianza posterior es menor.

In [132…
```python
pca2= PCA(n_components=6).fit(df7)
```

In [133…
```python
df8= pd.DataFrame(pca2.transform(df7),columns=["componente principal 1", "componente principal 2", "componente
                                              "componente principal 4", "componente principal 5","componente p
df8.head(10)
```

Out[133…

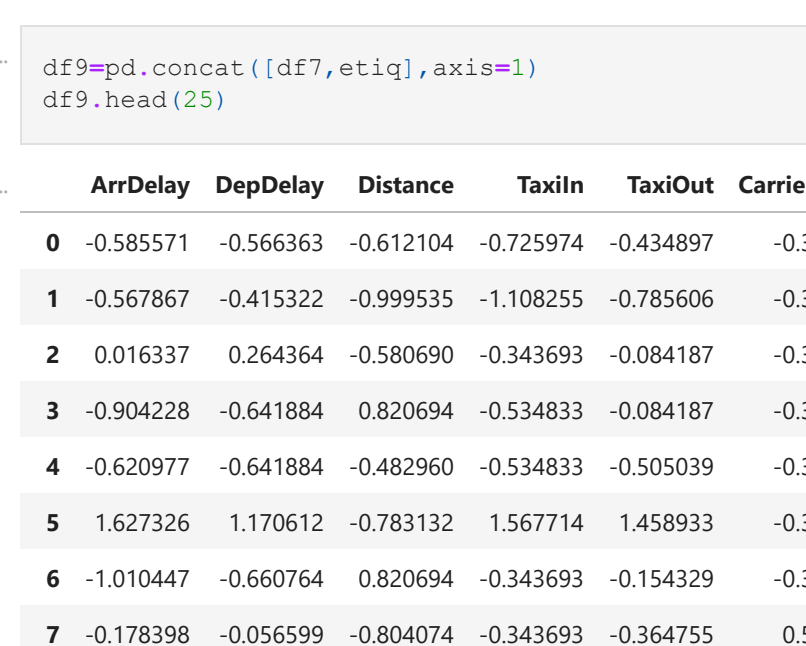
	componente principal 1	componente principal 2	componente principal 3	componente principal 4	componente principal 5	componente principal 6
0	-1.404389	0.499668	-0.200687	0.606562	-0.000784	-0.379354
1	-1.320299	0.933093	-0.361798	0.910557	0.006438	-0.616226
2	0.244115	0.175648	-0.748205	1.873641	0.368914	1.488720
3	-1.702335	0.027781	0.322780	-0.226882	0.044315	0.433396
4	-1.497698	0.437103	-0.151547	0.451421	-0.011677	-0.261647
5	3.394304	-2.604662	-0.898049	-0.295013	-0.280297	-1.138916
6	-1.805348	0.003431	0.327186	-0.313384	0.028573	0.483568
7	-0.336218	0.697906	0.370438	0.705731	-0.046024	-0.506043
8	-0.787232	-0.639218	0.902419	-0.424572	0.056725	0.576889
9	0.030741	0.002087	-0.305363	-0.305538	-0.083695	-0.092187

1. **Ejercicio 1**: Agrupa los vuelos por KMeans

Primero de todos vamos a buscar el número de agrupaciones por el método del codo.

In [134…
```python
sse = []
for k in range(1, 13):
    model = KMeans(n_clusters=k, init="random", )
    model.fit(df8)
    sse.append(model.inertia_)

plt.plot(range(1, 13), sse)
plt.xticks(range(1, 13))
plt.xlabel("Número de agrupaciones")
plt.ylabel("SSE")
plt.show()
```



Con el gráfico tal cual tengo dudas si el número agrupaciones debe ser 2, 3 o 5, así que voy a buscarlo de otra manera

In [135…
```python
from kneed import KneeLocator
loccodo = KneeLocator(range(1, 13), sse, curve="convex", direction="decreasing" )

loccodo.elbow
```

Out[135…
```
3
```

Una vez conocidos los agrupamientos que necesitamos, vamos a transformar el Data Set

In [136…
```python
kmeans= KMeans(n_clusters=3, init="random",n_init=10,max_iter=1000,random_state=57  )
kmeans.fit(df8)
etiquetas= kmeans.predict(df8)
```

In [137…
```python
etiq= pd.DataFrame(etiquetas, columns=["clase"])
etiq.value_counts()
```

Out[137…
```
clase
1        294678
2         80303
0         13376
dtype: int64
```

In [138…
```python
df9=pd.concat([df7,etiq],axis=1)
df9.head(25)
```

Out[138…

	ArrDelay	DepDelay	Distance	TaxiIn	TaxiOut	CarrierDelay	WeatherDelay	NASDelay	SecurityDelay	LateAircraftDelay	X10
0	-0.585571	-0.566363	-0.612104	-0.725974	-0.434897	-0.343256	-0.138081	-0.346468	-0.036093	-0.456058	-0.554644
1	-0.567867	-0.415322	-0.999535	-1.108255	-0.785606	-0.343256	-0.138081	-0.346468	-0.036093	-0.456058	-0.538045
2	0.016337	0.264364	-0.580690	-0.343693	-0.084187	-0.343256	2.358603	-0.346468	-0.036093	-0.456058	0.009715
3	-0.904228	-0.641884	0.820694	-0.534833	-0.084187	-0.343256	-0.138081	-0.346468	-0.036093	-0.456058	-0.853422
4	-0.620977	-0.641884	-0.482960	-0.534833	-0.505039	-0.343256	-0.138081	-0.346468	-0.036093	-0.456058	-0.587841
5	1.627326	1.170612	-0.783132	1.567714	1.458933	-0.343256	-0.138081	2.755455	-0.036093	0.860013	1.520204
6	-1.010447	-0.660764	0.820694	-0.343693	-0.154329	-0.343256	-0.138081	-0.346468	-0.036093	-0.456058	-0.953015
7	-0.178398	-0.056599	-0.804074	-0.343693	-0.364755	0.543950	-0.138081	-0.346468	-0.036093	-0.456058	-0.172872
8	-0.320023	-0.490842	1.438490	-0.343693	0.757515	0.128072	-0.138081	-0.096888	-0.036093	-0.456058	-0.305662
9	0.122556	-0.132119	0.038851	0.038589	-0.154329	-0.176905	-0.138081	0.117038	-0.036093	0.383987	0.109307
10	0.246479	0.283244	0.098187	-0.725974	-0.154329	-0.343256	-0.138081	-0.346468	-0.036093	1.112026	0.225499
11	-0.284617	-0.113239	-0.079822	0.229729	-0.294613	-0.121455	-0.138081	-0.346468	-0.036093	0.047969	-0.272465
12	0.175666	-0.075479	0.892247	-0.152552	-0.224471	-0.343256	-0.138081	0.117038	-0.036093	0.636001	0.159104
13	0.866090	0.755248	-0.072841	-0.917115	-0.224471	-0.343256	-0.138081	0.295309	-0.036093	1.588051	0.806456
14	-0.196101	-0.056599	0.126110	-0.725974	0.056096	-0.204630	-0.138081	-0.346468	-0.036093	0.271981	-0.189471
15	-0.674086	-0.528603	-0.950670	-0.725974	-0.855748	-0.343256	-0.138081	-0.346468	-0.036093	-0.456058	-0.637638
16	0.795277	1.151732	0.176721	0.229729	0.126238	2.068836	-0.138081	-0.346468	-0.036093	-0.456058	0.740061
17	0.458917	0.547566	-0.374758	-0.343693	-0.785606	-0.343256	-0.138081	-0.346468	-0.036093	1.448044	0.424684
18	1.839764	1.302773	-0.078077	-0.534833	-0.294613	2.761965	-0.138081	0.865778	-0.036093	-0.456058	1.719389
19	0.175666	0.377645	3.632189	0.038589	0.196380	0.322148	-0.138081	-0.346468	-0.036093	0.327984	0.159104
20	3.362237	3.851595	-0.460272	0.612011	-0.154329	6.088988	-0.138081	-0.346468	-0.036093	-0.456058	3.146884
21	3.061283	3.436231	1.455942	-0.534833	0.336664	5.617660	-0.138081	-0.346468	-0.036093	-0.456058	2.864705
22	-0.320023	-0.623004	-0.816290	-0.534833	0.617231	0.322148	-0.138081	-0.346468	-0.036093	-0.456058	-0.305662
23	-0.957337	-0.660764	1.192419	-0.343693	-0.925890	-0.343256	-0.138081	-0.346468	-0.036093	-0.456058	-0.903218
24	-0.373133	-0.415322	-0.996045	-0.152552	0.126238	-0.343256	-0.138081	-0.346468	-0.036093	0.131974	-0.355458

In [139…
```python
indice = ['ArrDelay', 'DepDelay', 'Distance', 'TaxiIn', 'TaxiOut', 'CarrierDelay',
       'WeatherDelay', 'NASDelay', 'SecurityDelay', 'LateAircraftDelay', 'X10',
       'X11']

for l in indice :
    sns.displot(df9, x=l, col="clase")
```

In [140…

```
# contando la proporción de miembros de Clase
etiq.value_counts()
```

Out[140…

```
clase
1        294678
2         80303
0         13376
dtype: int64
```

1. Podemos concluir que uno de los motivos de clasificación han sido ArrDelay, DepDelay, X10 y X11, ya que meintras el resto de variable tienen los mismos dominios para cada clase , las variables ArrDelay, DepDelay, X10 y X11 tienen dominios claramente diferenciados en función de la clase a la que pertence

In [141…

```
# miremos el describe() tras estandarizar el DataSet, para ver juego ha tenido la distancia y posibles Outlayer
pd.DataFrame(scaler.transform(df6), columns=df6.columns).describe()
```

Out[141…

	ArrDelay	DepDelay	Distance	TaxiIn	TaxiOut	CarrierDelay	WeatherDelay	NASDelay	SecurityDelay
count	3.883570e+05	3.883570e+05	3.883570e+05	3.883570e+05	3.883570e+05	3.883570e+05	3.883570e+05	3.883570e+05	3.883570e+05
mean	-7.859829e-17	-2.830113e-16	-5.150073e-18	-9.691159e-16	-3.660525e-16	-2.534906e-15	4.795082e-16	2.503332e-15	-3.644056e-15
std	1.000001e+00	1.000001e+00	1.000001e+00	1.000001e+00	1.000001e+00	1.000001e+00	1.000001e+00	1.000001e+00	1.000001e+00
min	-1.984121e+00	-6.985243e-01	-1.282255e+00	-1.299396e+00	-1.276599e+00	-3.432562e-01	-1.380806e-01	-3.464678e-01	-3.609314e-02
25%	-5.855706e-01	-5.852433e-01	-7.447377e-01	-5.348334e-01	-5.751804e-01	-3.432562e-01	-1.380806e-01	-3.464678e-01	-3.609314e-02
50%	-3.200230e-01	-3.586814e-01	-2.770278e-01	-1.525520e-01	-2.946130e-01	-3.432562e-01	-1.380806e-01	-3.464678e-01	-3.609314e-02
75%	2.464785e-01	1.888433e-01	4.053400e-01	2.297294e-01	1.963799e-01	-6.600426e-02	-1.380806e-01	-1.325420e-01	-3.609314e-02
max	4.268098e+01	4.576555e+01	7.325003e+00	3.692874e+01	2.425503e+01	6.719531e+01	6.651756e+01	3.562871e+01	1.492125e+02

Podemos observar que lo Outlayers no son excesivamente grandes

vamos a compar también las componentes principales.

In [143…

```
df10 = pd.concat([df9, df8], axis=1)
df10.head()
```

Out[143…

	ArrDelay	DepDelay	Distance	TaxiIn	TaxiOut	CarrierDelay	WeatherDelay	NASDelay	SecurityDelay	LateAircraftDelay	X10	
0	-0.585571	-0.566363	-0.612104	-0.725974	-0.434897	-0.343256	-0.138081	-0.346468	-0.036093	-0.456058	-0.554644	-0.
1	-0.567867	-0.415322	-0.999535	-1.108255	-0.785606	-0.343256	-0.138081	-0.346468	-0.036093	-0.456058	-0.538045	-0.
2	0.016337	0.264364	-0.580690	-0.343693	-0.084187	-0.343256	2.358603	-0.346468	-0.036093	-0.456058	0.009715	0.
3	-0.904228	-0.641884	0.820694	-0.534833	-0.084187	-0.343256	-0.138081	-0.346468	-0.036093	-0.456058	-0.853422	-0.
4	-0.620977	-0.641884	-0.482960	-0.534833	-0.505039	-0.343256	-0.138081	-0.346468	-0.036093	-0.456058	-0.587841	-0.

In [145…

```
indice2= ["componente principal 1" , "componente principal 2","componente principal 3", "componente principal 4
        "componente principal 5", "componente principal 6"]
for s in indice2 :
    sns.displot(df10, x=s, col="clase")
```


Podemos observar que la variable que determina la clasificación es la **Componente principal 1**, ya que su dominio varía en función de la clase. Vamos a ver como son las combinaciones lineales de las variables para sacar los componentes principales

In [147…

```
pd.DataFrame( data    = pca2.components_,columns = df7.columns,
    index   = ["componente principal 1" , "componente principal 2","componente principal 3", "componente princi
          "componente principal 5", "componente principal 6"])
```

Out[147…

	ArrDelay	DepDelay	Distance	TaxiIn	TaxiOut	CarrierDelay	WeatherDelay	NASDelay	SecurityDelay	LateAircraftDelay	
componente principal 1	0.455679	0.446139	-0.006972	0.072683	0.117151	0.251775	0.127080	0.193445	0.003756	0.261358	0.
componente principal 2	-0.028414	0.144431	-0.137661	-0.346136	-0.585387	0.250336	-0.074603	-0.603122	-0.000460	0.208334	-0.
componente principal 3	0.000169	0.007134	0.351833	0.035813	0.035670	0.697176	-0.236844	-0.013206	0.005811	-0.575532	-0.
componente principal 4	0.019147	-0.015667	-0.563940	-0.379466	0.157540	0.144160	0.553338	0.032690	-0.112705	-0.413890	0.
componente principal 5	0.003667	0.006027	0.038624	-0.069967	0.018660	-0.024566	0.149734	-0.034522	0.982508	-0.060169	0.
componente principal 6	-0.022855	0.029005	0.570599	0.193772	-0.142804	-0.107089	0.721442	-0.244474	-0.131605	-0.069748	-0.

Conclusiones. Tras hacer las agrupaciones con Kmeans podemos decir que :

a)Podemos determinar por observación que si la componente principal 1 (CP1) está en el rango CP1>5 pertence a la clase 0, si CP1<0 pertence a clase 1, y si 0>CP1>5 , pertence a la clase 2

b) la clase principal 1, es una combinación lineal de las variable del DataSet. Donde el coeficiente constante de ArrDelay, DepDelay, X10 y X11 valen aproximadamente 0.45,

c) estas 4 variables tienen una alta correlación,

In [150…

```
df7[["ArrDelay","DepDelay", "X10","X11"]].corr()
```

Out[150…

	ArrDelay	DepDelay	X10	X11
ArrDelay	1.000000	0.952835	0.925995	0.951128
DepDelay	0.952835	1.000000	0.884185	0.998096
X10	0.925995	0.884185	1.000000	0.885914
X11	0.951128	0.998096	0.885914	1.000000

d) Así que es posible que la PCA las haya comprido en la componente principal 1, dándoles poco peso en las otras 5 componentes principales. Dando explicación porque estás variables son las afectados en su dominio, en función de la clase. Clasificando los grupos en función de ArrDelay(o "DepDelay", "X10","X11", ya que están muy correlacionadas.)

In [ ]:

In [ ]:

In [ ]: