Bicicletas: simulacro de prueba de laboratorio

Antes de empezar, ejecutaremos la siguiente celda con las importaciones necesarias para poder realizar el ejercicio:

In [ ]:
```
import csv
from collections import namedtuple
from matplotlib import pylab as plt
```

1. Carga de datos (0.75 puntos)

Tomaremos los datos de un fichero de entrada llamado `bicicletas.csv` en el que se encuentra registrado el uso de un sistema de alquiler de bicicletas. El fichero se encuentra en la carpeta `./csv`. Cada línea del fichero de entrada contiene siete informaciones relativas a la fecha, hora, condiciones meteorológicas y número de bicicletas alquiladas:

- mes: de 1 a 12
- dia: desde 0 (domingo) hasta 6 (sábado)
- hora: de 0 a 23
- temperatura: temperatura del aire
- humedad: humedad relativa
- viento: velocidad del viento
- alquiladas: número de bicicletas alquiladas

He aquí un fragmento con las primeras líneas del fichero de entrada:

```
mes,dia,hora,temperatura,humedad,viento,alquiladas
1,6,0,0.24,0.81,0.0,16
1,6,1,0.22,0.8,0.0,40
1,6,2,0.22,0.8,0.0,32
1,6,3,0.24,0.75,0.0,13
1,6,4,0.24,0.75,0.0,1
```

La primera función que implementaremos será la de lectura. Será la encargada de leer los datos del fichero de entrada y cargarlos en una lista de tuplas:

In [ ]:
```
Registro = namedtuple('Registro', 'mes dia hora temperatura humedad v
iento alquiladas')

def lee_bicicletas(fichero):
    ''' Lee el fichero de entrada y devuelve una lista de tuplas

    ENTRADA:
       - fichero: nombre del fichero de entrada
    SALIDA:
       - lista de registros -> [Registro(int, int, int, float, float,
 float, int)]

    Cada línea del fichero de entrada contiene siete informaciones
    relativas a la fecha, hora, condiciones meteorológicas y número
    de bicicletas alquiladas:
       - mes: de 1 a 12
       - dia: de 0 (domingo) a 6 (sábado)
       - hora: de 0 a 23
       - temperatura: temperatura del aire
       - humedad: humedad relativa
       - viento: velocidad del viento
       - alquiladas: número de bicicletas alquiladas
    Hay que transformar ciertos elementos de la entrada en valores nu
méricos
    para que puedan ser procesados posteriormente.
    '''
    pass
```

In [ ]:
```
# Test de la función lee_biciletas
registros = lee_bicicletas('./csv/bicicletas.csv')

# La salida esperada de la siguiente instrucción es:
#    17379 [Registro(mes=1, dia=6, hora=0, temperatura=0.24, humedad=
0.81, viento=0.0, alquiladas=16),
#           Registro(mes=1, dia=6, hora=1, temperatura=0.22, humedad=
0.8, viento=0.0, alquiladas=40)]
print(len(registros), registros[:2])
```

2. Consulta y filtrado (7.25 puntos)

Una vez que hemos cargado los datos en una estructura en memoria ya podemos empezar a procesarlos. En esta sección implementaremos algunas funciones de consulta y filtrado que nos permitirán trabajar con ellos.

La primera función que implementaremos se llama `proporcion_fin_de_semana`. La función toma una lista de tuplas de registros y calcula qué proporción de bicicletas se alquila durante los fines de semana:

In [ ]:
```
def proporcion_fin_de_semana(registros):
    ''' Proporcion de bicicletas alquiladas los fines de semana

    ENTRADA:
       - registros: lista de registros -> [Registro(int, int, int, fl
oat, float, float, int)]
    SALIDA:
       - proporcion de bicicletas alquiladas -> float

    Toma como entrada una lista de tuplas de registros y calcula qué
 proporción
    de bicicletas se alquila durante los fines de semana:
    '''
    pass
```

In [ ]:
```
# Test de la función proporcion_fin_de_semana
proporcion = proporcion_fin_de_semana(registros)

# La salida esperada de la siguiente instrucción es:
#   0.27996473388386783
print(proporcion)
```

La segunda función se llama `filtra_por_meses`. Toma una lista de registros y una lista de meses, y selecciona solo los registros de los meses indicados:

In [ ]:
```
def filtra_por_meses(registros, meses):
    ''' Selecciona registros por meses

    ENTRADA:
       - registros: lista de registros -> [Registro(int, int, int, fl
oat, float, float, int)]
       - meses: lista de meses a seleccionar -> [int]
    SALIDA:
       - registros seleccionados -> [Registro(int, int, int, float, f
loat, float, int)]

    Toma una lista de registros y una lista de meses, y selecciona
    solo los registros de los meses indicados
    '''
    pass
```

In [ ]:
```
# Test de la función filtra_por_meses
filtrados = filtra_por_meses(registros, [2,3,4])

# La salida esperada de la siguiente instrucción es:
# 4251 [Registro(mes=2, dia=2, hora=0, temperatura=0.16, humedad=0.6
4, viento=0.1045, alquiladas=8),
#       Registro(mes=2, dia=2, hora=1, temperatura=0.16, humedad=0.6
9, viento=0.1045, alquiladas=3)]
print(len(filtrados), filtrados[:2])
```

La última función de esta sección se llama `agrupa_por_dias`. Toma como entrada una lista de registros, y produce como salida un diccionario cuyas claves son los días. Los valores del diccionario son las listas de los registros correspondientes a cada día.

In [ ]:
```
def agrupa_por_dias(registros):
    ''' Crea un diccionario de registros indexado por días

    ENTRADA:
       - registros: lista de registros -> [Registro(int, int, int, fl
oat, float, float, int)]
    SALIDA:
       - diccionario con listas de registros por día -> {str: [Regist
ro(int, int, int, float, float, float, int)]}

    Toma como entrada una lista de registros, y produce como
    salida un diccionario cuyas claves son los días. Los valores
    del diccionario son listas de registros correspondientes
    a cada día.

    La solución debe ser genérica y adaptarse a los datos que
    se reciben como parámetro para calcular el conjunto de claves del
    diccionario.
    '''
    pass
```

In [ ]:
```
# Test de la función agrupa_por_dias
grupos = agrupa_por_dias(registros)

# La salida esperada de la siguiente instrucción es:
# 0 2502 [Registro(mes=1, dia=0, hora=0, temperatura=0.46, humedad=0.
88, viento=0.2985, alquiladas=17)]
# 1 2479 [Registro(mes=1, dia=1, hora=0, temperatura=0.22, humedad=0.
44, viento=0.3582, alquiladas=5)]
# 2 2453 [Registro(mes=1, dia=2, hora=0, temperatura=0.16, humedad=0.
55, viento=0.1045, alquiladas=5)]
# 3 2475 [Registro(mes=1, dia=3, hora=0, temperatura=0.2, humedad=0.6
4, viento=0.0, alquiladas=6)]
# 4 2471 [Registro(mes=1, dia=4, hora=0, temperatura=0.18, humedad=0.
55, viento=0.0, alquiladas=11)]
# 5 2487 [Registro(mes=1, dia=5, hora=0, temperatura=0.2, humedad=0.6
4, viento=0.19399999999999998, alquiladas=17)]
# 6 2512 [Registro(mes=1, dia=6, hora=0, temperatura=0.24, humedad=0.
81, viento=0.0, alquiladas=16)]

for dia in grupos:
    print(dia, len(grupos[dia]), grupos[dia][:1])
```

3. Visualización (2 puntos)

La función de visualización que implementaremos será `muestra_distribucion_dias`. Toma como entrada una lista de registros, y genera un diagrama de barras en el que cada barra corresponde al número total de bicicletas alquiladas un día de la semana.

In [ ]:
```
def muestra_distribucion_dias(registros):
    ''' Genera una diagrama de barras con la distribución por días de l número de
        bicicletas alquiladas

        ENTRADA:
           - registros: lista de registros -> [Registro(int, int, in t, float, float, float, int)]
        SALIDA EN PANTALLA:
           - diagrama de barras con el número total de bicicletas alq uiladas cada día de la semana

        Se usarán las siguientes instrucciones matplotlib para genera r el diagrama
        de barras:
            plt.bar(range(len(nombres_dias)), conteos_dias, tick_labe l=nombres_dias)
            plt.show()

        Donde las variables significan lo siguiente:
            - nombres_dias: lista con los nombres de los días
            - conteos_dias: lista con el número de bicicletas alquila das cada día
    '''
    pass
```

La salida de la celda de test debería ser la siguiente:

In [ ]:
```
# Test de la función muestra_distribucion_dias
muestra_distribucion_dias(registros)
```