Desafío - Conceptos previos a Big Data

- Para realizar este desafío debes haber estudiado el material disponibilizado de la unidad.
- Crea una carpeta de trabajo y guarda todos los archivos correspondientes (notebook).
- Una vez terminade el desafío, comprime la carpeta y sube el `.zip` a la sección correspondiente.

Ejercicio 1: Generación Artifical de Datos

- A continuación se define la función `create_random_row`, la cual generará un registro artificial de un cliente en una compañía de seguros:

```
import random

def create_random_row():
    # simulamos la columna edad
    age = random.randint(18, 90)
    # simulamos la columna ingreso
    income = random.randrange(10000, 1000000, step=1000)
    # simulamos la situación laboral
    employment_status = random.choice(['Unemployed', 'Employed'])
    # simulamos si es que tiene deuda o no
    debt_status = random.choice(['Debt', 'No Debt'])
    # simulamos si es que se cambió recientemente o no
    churn_status = random.choice(['Churn', 'No Churn'])

    return age, income, employment_status, use_in_minutes, debt_status,
churn_status

# ejecución
create_random_row()
# retorno
(85, 855000, 'Employed', 102.556, 'Debt', 'No Churn')
```

- Replique la función 10 millones de veces y preservela en un objeto.

Algunos supuestos:

- Asuma, de aquí en adelante, que los datos generados representarán mediciones empíricas sobre el comportamiento de clientes en la compañía de seguros.
- Considere el siguiente ambiente de trabajo de su computador: No tiene instalada la distribución `anaconda`, por lo que no tendrá acceso a las librerías `pandas`, `numpy` y `scipy`. Tampoco tiene permisos de usuario, por lo cual no podrá instalarlas. **Sólo puede implementar funciones nativas de Python.**

- Dado que su código podrá ser utilizado posteriormente en una aplicación web de uso interno montada en Scala, **debe utilizar operaciones vectorizadas como `map`, `filter`, `reduce`; y comprensiones de lista**.

Ejercicio 2:

Desde la gerencia de estudios de la compañía de seguros, le solicitan mejorar la siguiente línea de código:

```
employment_income_looped = 0

for i in random_database:
    if i[2] == 'Employed':
        employment_income_looped += i[1]


# retorno
2523162067000
```

Responda los siguientes puntos:

- ¿Qué retornará la variable `employment_income_looped`?
- ¿Cómo sería una implementación del código utilizando `map` y `filter`?
- ¿Son iguales los resultados?

Ejercicio 3:

Desde la gerencia le solicitan mejorar la siguiente línea de código:

```
count_debts_looped = 0

for i in random_database:
    for j in i:
        if j == 'Debt':
            count_debts_looped += 1

# retorno
5000335
```

Responda los siguientes puntos:

- ¿Cuál será el retorno de la variable `count_debts_looped`?
- ¿Cuál es la complejidad algorítmica del código?
- ¿Cómo sería una implementación del código utilizando `map` y `filter`?
- ¿Son iguales los resultados de ambas operaciones?

Ejercicio 4

Desde la gerencia le solicitan mejorar la siguiente línea de código:

```
churn_subset, no_churn_subset = [], []

for i in random_database:
    for j in i:
        if i == 'Churn':
            churn_subset.append(i)
    for j in i:
        if i == 'No Churn':
            no_churn.append(i)
```

- ¿Cuál será el retorno de la variable `churn_subset` y `no_churn_subset`?
- ¿Cuál es la complejidad algorítmica del código?
- ¿Cómo sería una implementación del código utilizando `map` y `filter`?
- ¿Son iguales los resultados de ambas operaciones?
- Estime la media, la varianza, el mínimo y el máximo de la edad para ambos subsets, sin utilizar librerías externas.

Ejercicio 5:

Desde la gerencia le solicitan mejorar la siguiente línea de código:

```
unemployed_debt_churn = 0
unemployed_nodebt_churn = 0
unemployed_debt_nochurn = 0
unemployed_nodebt_nochurn = 0
employed_debt_churn = 0
employed_nodebt_churn = 0
employed_debt_nochurn = 0
employed_nodebt_nochurn = 0

for i in random_database:
    if i[2] == 'Unemployed' and i[3] == 'Debt' and i[4] == 'Churn':
        unemployed_debt_churn += 1
    if i[2] == 'Unemployed' and i[3] == 'No Debt' and i[4] == 'Churn':
        unemployed_nodebt_churn += 1
    if i[2] == 'Unemployed' and i[3] == 'Debt' and i[4] == 'No Churn':
        unemployed_debt_nochurn += 1
    if i[2] == 'Unemployed' and i[3] == 'No Debt' and i[4] == 'No Churn':
        unemployed_nodebt_nochurn += 1
    if i[2] == 'Employed' and i[3] == 'Debt' and i[4] == 'Churn':
        employed_debt_churn += 1
    if i[2] == 'Employed' and i[3] == 'No Debt' and i[4] == 'Churn':
        employed_nodebt_churn += 1
    if i[2] == 'Employed' and i[3] == 'Debt' and i[4] == 'No Churn':
        employed_debt_nochurn += 1
    if i[2] == 'Employed' and i[3] == 'No Debt' and i[4] == 'No Churn':
        employed_nodebt_nochurn += 1

print("Unemployed, Debt, Churn: ", unemployed_debt_churn)
print("Unemployed, No Debt, Churn: ", unemployed_nodebt_churn)
print("Unemployed, Debt, No Churn: ", unemployed_debt_nochurn)
print("Unemployed, No Debt, No Churn: ", unemployed_nodebt_nochurn)
print("Employed, Debt, Churn: ", employed_debt_churn)
print("Employed, No Debt, Churn: ", employed_nodebt_churn)
print("Employed, Debt, No Churn: ", employed_debt_nochurn)
print("Employed, No Debt, No Churn: ", employed_nodebt_nochurn)


# retorno
Unemployed, Debt, Churn:  1249114
Unemployed, No Debt, Churn:  1250165
Unemployed, Debt, No Churn:  1251163
Unemployed, No Debt, No Churn:  1249760
Employed, Debt, Churn:  1249421
Employed, No Debt, Churn:  1250581
Employed, Debt, No Churn:  1248184
Employed, No Debt, No Churn:  1251612
```

- ¿Cómo sería una implementación utilizando `map`?
- ¿Son iguales los resultados de ambas operaciones?