# P1_Algebra lineal

July 24, 2024

---

## 1 Descripción

La compañía de seguros Sure Tomorrow quiere resolver varias tareas con la ayuda de machine learning y te pide que evalúes esa posibilidad. - Tarea 1: encontrar clientes que sean similares a un cliente determinado. Esto ayudará a los agentes de la compañía con el marketing. - Tarea 2: predecir la probabilidad de que un nuevo cliente reciba una prestación del seguro. ¿Puede un modelo de predictivo funcionar mejor que un modelo dummy? - Tarea 3: predecir el número de prestaciones de seguro que un nuevo cliente pueda recibir utilizando un modelo de regresión lineal. - Tarea 4: proteger los datos personales de los clientes sin afectar al modelo del ejercicio anterior. Es necesario desarrollar un algoritmo de transformación de datos que dificulte la recuperación de la información personal si los datos caen en manos equivocadas. Esto se denomina enmascaramiento u ofuscación de datos. Pero los datos deben protegerse de tal manera que no se vea afectada la calidad de los modelos de machine learning. No es necesario elegir el mejor modelo, basta con demostrar que el algoritmo funciona correctamente.

## 2 Preprocesamiento y exploración de datos

### 2.1 Inicialización

[1]:
```
pip install scikit-learn --upgrade
```

```
Requirement already satisfied: scikit-learn in
/opt/conda/envs/python3/lib/python3.9/site-packages (0.24.1)
Collecting scikit-learn
  Downloading scikit_learn-1.5.0-cp39-cp39-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (11 kB)
Requirement already satisfied: numpy>=1.19.5 in
/opt/conda/envs/python3/lib/python3.9/site-packages (from scikit-learn) (1.21.2)
Requirement already satisfied: scipy>=1.6.0 in
/opt/conda/envs/python3/lib/python3.9/site-packages (from scikit-learn) (1.10.1)
Requirement already satisfied: joblib>=1.2.0 in
/opt/conda/envs/python3/lib/python3.9/site-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in
/opt/conda/envs/python3/lib/python3.9/site-packages (from scikit-learn) (3.5.0)
Downloading
scikit_learn-1.5.0-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
```

```
(13.4 MB)
                                    13.4/13.4 MB
64.5 MB/s eta 0:00:00:00:0100:01
Installing collected packages: scikit-learn
  Attempting uninstall: scikit-learn
    Found existing installation: scikit-learn 0.24.1
    Uninstalling scikit-learn-0.24.1:
ERROR: Could not install packages due to an OSError: [Errno 13] Permission

denied: 'COPYING'

Consider using the `--user` option or check the permissions.


Note: you may need to restart the kernel to use updated packages.
```

[2]:
```
import numpy as np
import pandas as pd

import seaborn as sns

import sklearn.linear_model
import sklearn.metrics
import sklearn.neighbors
import sklearn.preprocessing

from sklearn.model_selection import train_test_split

from IPython.display import display

from sklearn.neighbors import NearestNeighbors, KNeighborsClassifier
import math
```

### 2.2 Carga de datos

Carga los datos y haz una revisión básica para comprobar que no hay problemas obvios.

[3]:
```
df = pd.read_csv('/datasets/insurance_us.csv')
```

Renombramos las columnas para que el código se vea más coherente con su estilo.

[4]:
```
df = df.rename(columns={'Gender': 'gender', 'Age': 'age', 'Salary': 'income',␣
 ↪'Family members': 'family_members', 'Insurance benefits':␣
 ↪'insurance_benefits'})
```

[5]:
```
df.sample(10)
```

[5]:

| | gender | age | income | family_members | insurance_benefits |
|---|---|---|---|---|---|
| 1370 | 1 | 33.0 | 42400.0 | 1 | 0 |

```
3994       1  25.0  55500.0              1                    0
1316       0  35.0  21200.0              0                    0
4005       1  29.0  37200.0              1                    0
366        1  34.0  46300.0              2                    0
670        1  28.0  34600.0              1                    0
487        0  30.0  62000.0              0                    0
1854       1  19.0  58100.0              1                    0
2522       1  19.0  46200.0              1                    0
1583       0  32.0  50500.0              0                    0
```

[6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 5 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   gender              5000 non-null   int64
 1   age                 5000 non-null   float64
 2   income              5000 non-null   float64
 3   family_members      5000 non-null   int64
 4   insurance_benefits  5000 non-null   int64
dtypes: float64(2), int64(3)
memory usage: 195.4 KB
```

[7]:
```
# puede que queramos cambiar el tipo de edad (de float a int) aunque esto no es␣
↪crucial

# escribe tu conversión aquí si lo deseas:
df['age'] = df['age'].astype(int)
df.head()
```

[7]:

| | gender | age | income | family_members | insurance_benefits |
|---|---|---|---|---|---|
| 0 | 1 | 41 | 49600.0 | 1 | 0 |
| 1 | 0 | 46 | 38000.0 | 1 | 1 |
| 2 | 0 | 29 | 21000.0 | 0 | 0 |
| 3 | 0 | 21 | 41700.0 | 2 | 0 |
| 4 | 1 | 28 | 26100.0 | 0 | 0 |

[8]:
```
# comprueba que la conversión se haya realizado con éxito
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 5 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
```

```
 0   gender              5000 non-null   int64
 1   age                 5000 non-null   int64
 2   income              5000 non-null   float64
 3   family_members      5000 non-null   int64
 4   insurance_benefits  5000 non-null   int64
dtypes: float64(1), int64(4)
memory usage: 195.4 KB
```

[9]:
```
# ahora echa un vistazo a las estadísticas descriptivas de los datos.# ¿Se ve␣
 ↪todo bien?
df.describe(percentiles= np.linspace(0,1,11))
```

[9]:

| | gender | age | income | family_members | insurance_benefits |
|---|---|---|---|---|---|
| count | 5000.000000 | 5000.000000 | 5000.000000 | 5000.000000 | 5000.000000 |
| mean | 0.499000 | 30.952800 | 39916.360000 | 1.194200 | 0.148000 |
| std | 0.500049 | 8.440807 | 9900.083569 | 1.091387 | 0.463183 |
| min | 0.000000 | 18.000000 | 5300.000000 | 0.000000 | 0.000000 |
| 0% | 0.000000 | 18.000000 | 5300.000000 | 0.000000 | 0.000000 |
| 10% | 0.000000 | 20.000000 | 27200.000000 | 0.000000 | 0.000000 |
| 20% | 0.000000 | 23.000000 | 31700.000000 | 0.000000 | 0.000000 |
| 30% | 0.000000 | 25.000000 | 34700.000000 | 0.000000 | 0.000000 |
| 40% | 0.000000 | 28.000000 | 37400.000000 | 1.000000 | 0.000000 |
| 50% | 0.000000 | 30.000000 | 40200.000000 | 1.000000 | 0.000000 |
| 60% | 1.000000 | 33.000000 | 42500.000000 | 1.000000 | 0.000000 |
| 70% | 1.000000 | 35.000000 | 45100.000000 | 2.000000 | 0.000000 |
| 80% | 1.000000 | 38.000000 | 48200.000000 | 2.000000 | 0.000000 |
| 90% | 1.000000 | 43.000000 | 52500.000000 | 3.000000 | 1.000000 |
| 100% | 1.000000 | 65.000000 | 79000.000000 | 6.000000 | 5.000000 |
| max | 1.000000 | 65.000000 | 79000.000000 | 6.000000 | 5.000000 |

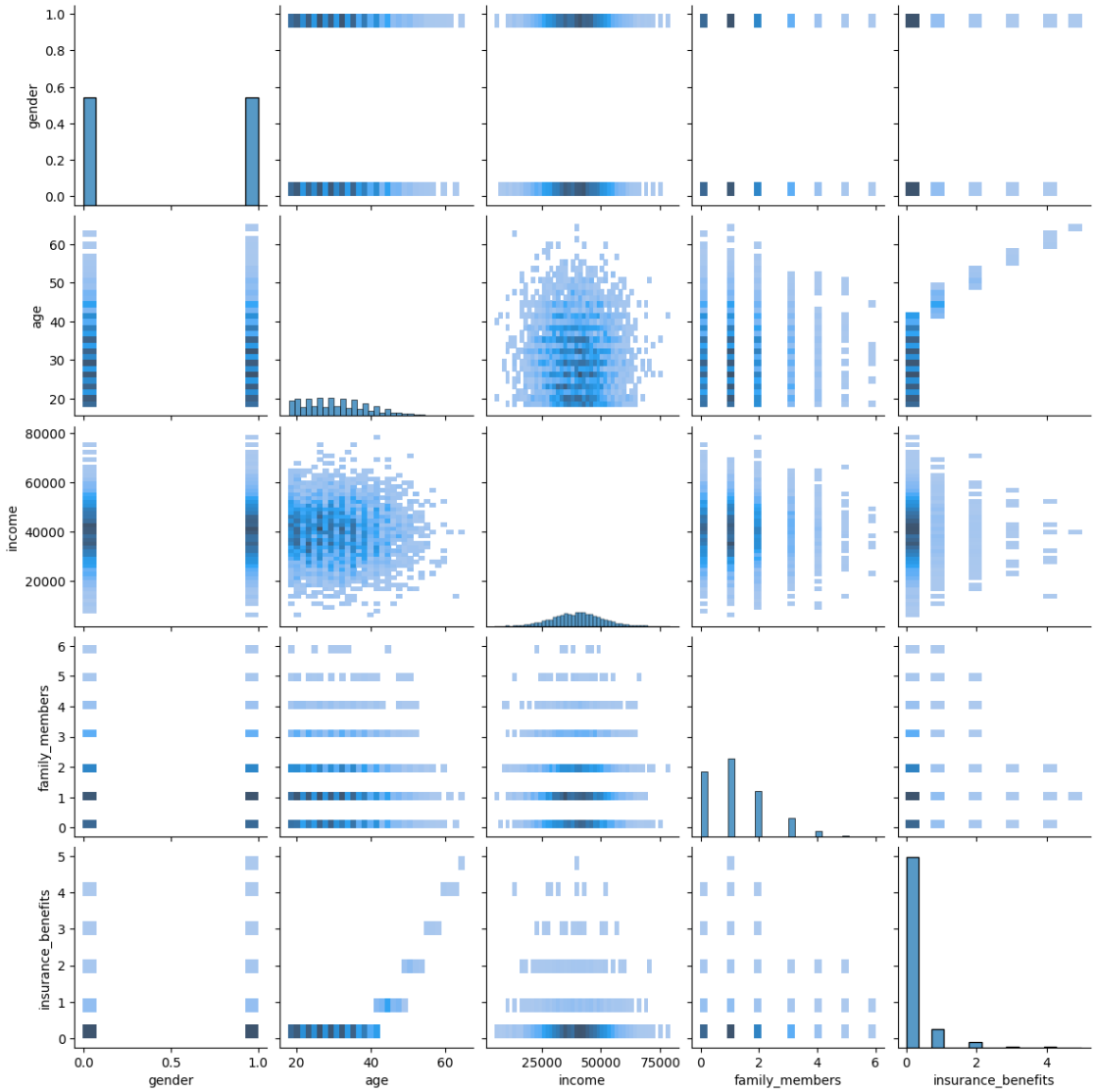
Podemos observar que:

- La edad media de nuestros clientes es de 30-31 años
- Se tiene una población equilibrada entre el género 0 y 1 (50.1% versus 49.9%)
- El ingreso medio es de 39916 dls
- La cantidad media de miembros de familia es de 1.19 miembros.
- el 80% de nuestros datos tiene un valor de 0 en la cantidad de beneficios recibidos por la aseguradora 'insurance_benefits'

## 2.3 Análisis exploratorio de datos

Vamos a comprobar rápidamente si existen determinados grupos de clientes observando el gráfico de pares.

[10]:
```
g = sns.pairplot(df, kind='hist')
g.fig.set_size_inches(12, 12)
```



De acuerdo, es un poco complicado detectar grupos obvios (clústeres) ya que es difícil combinar diversas variables simultáneamente (para analizar distribuciones multivariadas). Ahí es donde LA y ML pueden ser bastante útiles.

## 3 Tarea 1. Clientes similares

En el lenguaje de ML, es necesario desarrollar un procedimiento que devuelva los k vecinos más cercanos (objetos) para un objeto dado basándose en la distancia entre los objetos. Es posible que quieras revisar las siguientes lecciones (capítulo -> lección)- Distancia entre vectores -> Distancia euclidiana - Distancia entre vectores -> Distancia Manhattan

Para resolver la tarea, podemos probar diferentes métricas de distancia.

Escribe una función que devuelva los k vecinos más cercanos para un $n^{th}$ objeto basándose en una métrica de distancia especificada. A la hora de realizar esta tarea no debe tenerse en cuenta el número de prestaciones de seguro recibidas. Puedes utilizar una implementación ya existente del algoritmo kNN de scikit-learn (consulta el enlace) o tu propia implementación. Pruébalo para cuatro combinaciones de dos casos- Escalado - los datos no están escalados - los datos se escalan con el escalador MaxAbsScaler - Métricas de distancia - Euclidiana - Manhattan

Responde a estas preguntas:- ¿El hecho de que los datos no estén escalados afecta al algoritmo kNN? Si es así, ¿cómo se manifiesta?- ¿Qué tan similares son los resultados al utilizar la métrica de distancia Manhattan (independientemente del escalado)?

[11]:
```
feature_names = ['gender', 'age', 'income', 'family_members']
```

[12]:
```
def get_knn(df, n, k, metric):

    """
    Devuelve los k vecinos más cercanos

    :param df: DataFrame de pandas utilizado para encontrar objetos similares␣
↪dentro del mismo lugar
    :param n: número de objetos para los que se buscan los vecinos más cercanos␣
↪
    :param k: número de vecinos más cercanos a devolver
    :param métrica: nombre de la métrica de distancia    """

    nbrs = NearestNeighbors(n_neighbors=n , metric = metric )
    nbrs.fit(df)
    nbrs_distances, nbrs_indices = nbrs.kneighbors([df.iloc[n][feature_names]],␣
↪k, return_distance=True)

    df_res = pd.concat([
        df.iloc[nbrs_indices[0]],
```

```
        pd.DataFrame(nbrs_distances.T, index=nbrs_indices[0],␣
↪columns=['distance'])
        ], axis=1)

    return df_res
```

Escalar datos.

[13]:
```
feature_names = ['gender', 'age', 'income', 'family_members']

transformer_mas = sklearn.preprocessing.MaxAbsScaler().fit(df[feature_names].
↪to_numpy())

df_scaled = df.copy()
df_scaled.loc[:, feature_names] = transformer_mas.transform(df[feature_names].
↪to_numpy())
```

[14]:
```
df_scaled.sample(5)
```

[14]:

| | gender | age | income | family_members | insurance_benefits |
|---|---|---|---|---|---|
| 1147 | 1.0 | 0.646154 | 0.516456 | 0.333333 | 0 |
| 2772 | 1.0 | 0.292308 | 0.693671 | 0.166667 | 0 |
| 2652 | 0.0 | 0.476923 | 0.475949 | 0.000000 | 0 |
| 4622 | 0.0 | 0.292308 | 0.607595 | 0.000000 | 0 |
| 1298 | 0.0 | 0.646154 | 0.565823 | 0.166667 | 1 |

Ahora, vamos a obtener registros similares para uno determinado, para cada combinación

- los datos no están escalados
- los datos se escalan con el escalador MaxAbsScaler
- Métricas de distancia
  - Euclidiana
  - Manhattan

### 3.0.1 Datos no escalados

[15]:
```
#Datos no escalados con métrica distancia euclidiana
get_knn(df[feature_names],n=2,k=6, metric='euclidean' )
```

[15]:

| | gender | age | income | family_members | distance |
|---|---|---|---|---|---|
| 2 | 0 | 29 | 21000.0 | 0 | 0.000000 |
| 4544 | 1 | 31 | 21000.0 | 1 | 2.449490 |
| 2031 | 0 | 30 | 20900.0 | 0 | 100.005000 |
| 1741 | 1 | 28 | 21100.0 | 0 | 100.010000 |
| 4805 | 0 | 28 | 21100.0 | 2 | 100.024997 |
| 2752 | 1 | 27 | 21100.0 | 1 | 100.029996 |

[16]:
```
#Datos no escalados con métrica distancia Manhattan
answer = get_knn(df[feature_names],n=2,k=6, metric='cityblock' )
answer
```

[16]:

| | gender | age | income | family_members | distance |
|---|---|---|---|---|---|
| 2 | 0 | 29 | 21000.0 | 0 | 0.0 |
| 4544 | 1 | 31 | 21000.0 | 1 | 4.0 |
| 2031 | 0 | 30 | 20900.0 | 0 | 101.0 |
| 1741 | 1 | 28 | 21100.0 | 0 | 102.0 |
| 4805 | 0 | 28 | 21100.0 | 2 | 103.0 |
| 2752 | 1 | 27 | 21100.0 | 1 | 104.0 |

### 3.0.2 Datos escalados con MaxAbsScaler

[17]:
```
# datos escalados con MaxAbsScaler y métrica distancia Euclidiana
get_knn(df_scaled[feature_names],n=2 ,k=6,metric='euclidean')
```

[17]:

| | gender | age | income | family_members | distance |
|---|---|---|---|---|---|
| 2 | 0.0 | 0.446154 | 0.265823 | 0.0 | 0.000000 |
| 509 | 0.0 | 0.446154 | 0.255696 | 0.0 | 0.010127 |
| 224 | 0.0 | 0.446154 | 0.277215 | 0.0 | 0.011392 |
| 2031 | 0.0 | 0.461538 | 0.264557 | 0.0 | 0.015437 |
| 648 | 0.0 | 0.461538 | 0.289873 | 0.0 | 0.028550 |
| 3156 | 0.0 | 0.446154 | 0.308861 | 0.0 | 0.043038 |

[18]:
```
# datos escalados con MaxAbsScaler y métrica distancia Manhattan
answer2 = get_knn(df_scaled[feature_names],n=2 ,k=5+1,metric='cityblock')
answer2
```

[18]:

| | gender | age | income | family_members | distance |
|---|---|---|---|---|---|
| 2 | 0.0 | 0.446154 | 0.265823 | 0.0 | 0.000000 |
| 509 | 0.0 | 0.446154 | 0.255696 | 0.0 | 0.010127 |
| 224 | 0.0 | 0.446154 | 0.277215 | 0.0 | 0.011392 |
| 2031 | 0.0 | 0.461538 | 0.264557 | 0.0 | 0.016650 |
| 648 | 0.0 | 0.461538 | 0.289873 | 0.0 | 0.039435 |
| 3156 | 0.0 | 0.446154 | 0.308861 | 0.0 | 0.043038 |

[19]:
```
#comparando los índices de los datos escalados y no escalados para generar␣
 ↪conclusiones
print(df.iloc[answer.index])
print('-----------------------------------------------')
print(df.iloc[answer2.index])
```

| | gender | age | income | family_members | insurance_benefits |
|---|---|---|---|---|---|
| 2 | 0 | 29 | 21000.0 | 0 | 0 |
| 4544 | 1 | 31 | 21000.0 | 1 | 0 |
| 2031 | 0 | 30 | 20900.0 | 0 | 0 |

```
1741       1   28  21100.0              0                    0
4805       0   28  21100.0              2                    0
2752       1   27  21100.0              1                    0
-----------------------------------------------
      gender  age   income  family_members  insurance_benefits
2          0   29  21000.0              0                    0
509        0   29  20200.0              0                    0
224        0   29  21900.0              0                    0
2031       0   30  20900.0              0                    0
648        0   30  22900.0              0                    0
3156       0   29  24400.0              0                    0
```

Respuestas a las preguntas

**¿El hecho de que los datos no estén escalados afecta al algoritmo kNN? Si es así, ¿cómo se manifiesta?**

Si afecta, se manifiesta de la siguiente manera: * Podemos observar que answer2 (datos escalados) nos arroja una lista de elementos con una mayor cantidad de características en comun, por ejemplo, el elemento muestra (índice 2) tiene las siguientes características: * Gender = 0 * Age = 29 * Income = 21000 * Family_members = 0 * Insurance_benefits = 0

`La mayor parte de los vecinos propuestos por el modelo coinciden con las columnas anteriores, p`

Todo esto se debe a la sensibilidad de la escala, el modelo KNN calcula las distancias entre los puntos de datos para encontrar sus vecinos más cercanos, si tenemos una (o más) características con escalas muy distantes entre sí, las distancias de esta característica van a aportar más peso al resultado final que las restantes características.

Habiendo dicho esto, concluimos que los datos con mayor cercanía al vecino objetivo son los datos escalados.

**¿Qué tan similares son los resultados al utilizar la métrica de distancia Manhattan (independientemente del escalado)?**

Los datos son prácticamente idénticos con la métrica de distancia Manhattan, siempre y cuando se comparen con el mismo tipo de escalado (Con o sin escalado)

# 4 Tarea 2. ¿Es probable que el cliente reciba una prestación del seguro?

En términos de machine learning podemos considerarlo como una tarea de clasificación binaria.

Con el valor de `insurance_benefits` superior a cero como objetivo, evalúa si el enfoque de clasificación kNN puede funcionar mejor que el modelo dummy. Instrucciones: - Construye un clasificador basado en KNN y mide su calidad con la métrica F1 para k=1…10 tanto para los datos originales como para los escalados. Sería interesante observar cómo k puede influir en la métrica de evaluación y si el escalado de los datos provoca alguna diferencia. Puedes utilizar una implementación ya existente del algoritmo de clasificación kNN de scikit-learn (consulta el enlace) o tu propia implementación. - Construye un modelo dummy que, en este caso, es simplemente un modelo

aleatorio. Debería devolver "1" con cierta probabilidad. Probemos el modelo con cuatro valores de probabilidad: 0, la probabilidad de pagar cualquier prestación del seguro, 0.5, 1.

La probabilidad de pagar cualquier prestación del seguro puede definirse como

$$P\{\text{prestación de seguro recibida}\} = \frac{\text{número de clientes que han recibido alguna prestación de seguro}}{\text{número total de clientes}}.$$

Divide todos los datos correspondientes a las etapas de entrenamiento/prueba respetando la proporción 70:30.

[20]:
```
# alcula el objetivo
df['insurance_benefits_received'] = df['insurance_benefits']>0
```

[21]:
```
# comprueba el desequilibrio de clases con value_counts()

print(f"Probabilidad de clase 0 / False = { (round(((␣
 ↪(df['insurance_benefits']>0).value_counts()/df['insurance_benefits'].
 ↪count())[0]) *100, 2) ) } % ")
print(f"Probabilidad de clase 1 / True = { (round(((␣
 ↪(df['insurance_benefits']>0).value_counts()/df['insurance_benefits'].
 ↪count())[1]) *100, 2) ) } % ")
```

```
Probabilidad de clase 0 / False = 88.72 %
Probabilidad de clase 1 / True = 11.28 %
```

Podemos observar un claro desequilibrio de clases

[22]:
```
def eval_classifier(y_true, y_pred):

    f1_score = sklearn.metrics.f1_score(y_true, y_pred)
    print(f'F1: {f1_score:.2f}')

# si tienes algún problema con la siguiente línea, reinicia el kernel y ejecuta␣
 ↪el cuaderno de nuevo
    cm = sklearn.metrics.confusion_matrix(y_true, y_pred, normalize='all')
    print('Matriz de confusión')
    print(cm)
```

[23]:
```
# generar la salida de un modelo aleatorio

def rnd_model_predict(P, size, seed=42):

    rng = np.random.default_rng(seed=seed)
    return rng.binomial(n=1, p=P, size=size)
```

[24]:
```
for P in [0, df['insurance_benefits_received'].sum() / len(df), 0.5, 1]:
```

```
    print(f'La probabilidad: {P:.2f}')
    y_pred_rnd = rnd_model_predict(P, size= len(df))

    eval_classifier(df['insurance_benefits_received'], y_pred_rnd)

    print()
```

```
La probabilidad: 0.00
F1: 0.00
Matriz de confusión
[[0.8872 0.    ]
 [0.1128 0.    ]]

La probabilidad: 0.11
F1: 0.12
Matriz de confusión
[[0.7914 0.0958]
 [0.0994 0.0134]]

La probabilidad: 0.50
F1: 0.20
Matriz de confusión
[[0.456  0.4312]
 [0.053  0.0598]]

La probabilidad: 1.00
F1: 0.20
Matriz de confusión
[[0.     0.8872]
 [0.     0.1128]]
```

[25]:
```
#Re escalando datos para añadir la columna 'insurance_benefits_received' a␣
 ↪df_scaled
df_scaled = df.copy()
df_scaled.loc[:, feature_names] = transformer_mas.transform(df[feature_names].
 ↪to_numpy())
```

[26]:
```
#Dividiendo datos de validación y entrenamiento
df_train, df_test = train_test_split(df, test_size = 0.3, random_state=1234)
#Dividiendo datos de validación y entrenamiento  para el df escalado
df_train_scaled, df_test_scaled = train_test_split(df_scaled, test_size = 0.3,␣
 ↪random_state=1234)
```

[27]:
```
#Datos no escalados
best_f1_score = -1
best_k = -1
```

```
for k in range(1,11):

    neigh = KNeighborsClassifier(n_neighbors=k)
    neigh.fit(df_train[feature_names], df_train['insurance_benefits_received'])

    predicted_values1= neigh.predict(df_test[feature_names])
    # score= neigh.
 ↪score(df_test[feature_names],df_test['insurance_benefits_received'])
    # print(score)

    f1_score = sklearn.metrics.
 ↪f1_score(df_test['insurance_benefits_received'],pd.Series(predicted_values1))
    print(f"F1 Score para clasificador KNN con K= {k} : {round(f1_score,3)}")

    if f1_score > best_f1_score:
        best_f1_score = f1_score
        best_k = k

print('')
print(f"Mejor F1 score: {round(best_f1_score,4)}, mejor valor K: {best_k}")
# df_test['insurance_benefits_received'].unique()
# pd.Series(predicted_values1).unique()
```

```
F1 Score para clasificador KNN con K= 1 : 0.63
F1 Score para clasificador KNN con K= 2 : 0.377
F1 Score para clasificador KNN con K= 3 : 0.41
F1 Score para clasificador KNN con K= 4 : 0.212
F1 Score para clasificador KNN con K= 5 : 0.208
F1 Score para clasificador KNN con K= 6 : 0.108
F1 Score para clasificador KNN con K= 7 : 0.106
F1 Score para clasificador KNN con K= 8 : 0.045
F1 Score para clasificador KNN con K= 9 : 0.067
F1 Score para clasificador KNN con K= 10 : 0.034

Mejor F1 score: 0.6299, mejor valor K: 1
```

[28]:
```
#Datos escalados
best_f1_score = -1
best_k = -1
for k in range(1,11):

    neigh = KNeighborsClassifier(n_neighbors=k)
    neigh.fit(df_train_scaled[feature_names],␣
 ↪df_train_scaled['insurance_benefits_received'])

    predicted_values1= neigh.predict(df_test_scaled[feature_names])
```

```
    # score= neigh.
↪score(df_test[feature_names],df_test['insurance_benefits_received'])
    # print(score)

    f1_score = sklearn.metrics.
↪f1_score(df_test_scaled['insurance_benefits_received'],pd.
↪Series(predicted_values1))
    print(f"F1 Score para clasificador KNN con K= {k} : {round(f1_score,3)}")
    if f1_score > best_f1_score:
        best_f1_score = f1_score
        best_k = k

print('')
print(f"Mejor F1 score: {round(best_f1_score,4)}, mejor valor K: {best_k}")
```

```
F1 Score para clasificador KNN con K= 1 : 0.953
F1 Score para clasificador KNN con K= 2 : 0.915
F1 Score para clasificador KNN con K= 3 : 0.952
F1 Score para clasificador KNN con K= 4 : 0.929
F1 Score para clasificador KNN con K= 5 : 0.934
F1 Score para clasificador KNN con K= 6 : 0.926
F1 Score para clasificador KNN con K= 7 : 0.938
F1 Score para clasificador KNN con K= 8 : 0.929
F1 Score para clasificador KNN con K= 9 : 0.955
F1 Score para clasificador KNN con K= 10 : 0.919

Mejor F1 score: 0.955, mejor valor K: 9
```

### 4.0.1 Conclusiones

- El modelo dummy tuvo en su mejor desempeño, un F1 score de 0.2 presentado en las probabilidades de 0.5 y 1
- El modelo K Neighbors Classifier con las características *no escaladas* presentó un desempeño máximo con un valor K=9, donde el F1 Score fue de .6299, cabe mencionar que solamente 4 de 10 iteraciones de valores K fueron superiores al modelo dummy con F1 Score $> 0.2$
- **El modelo K Neighbors Classifier presentó el mejor desempeño con las características escaladas y un valor K = 9, donde el F1 Score fue de 0.955**

Podemos entonces concluir que el comportamiento del modelo de clasificación por K-N Vecinos es óptimo cuando los datos se escalan, también resaltamos la utilidad de realizar iteraciones de los K vecinos para verificar el mejor comportamiento.

## 5 Tarea 3. Regresión (con regresión lineal)

Con `insurance_benefits` como objetivo, evalúa cuál sería la RECM de un modelo de regresión lineal.

Construye tu propia implementación de regresión lineal. Para ello, recuerda cómo está formulada la solución de la tarea de regresión lineal en términos de LA. Comprueba la RECM tanto para los datos originales como para los escalados. ¿Puedes ver alguna diferencia en la RECM con respecto a estos dos casos?

Denotemos- $X$: matriz de características; cada fila es un caso, cada columna es una característica, la primera columna está formada por unidades - $y$ — objetivo (un vector) - $\hat{y}$ — objetivo estimado (un vector) - $w$ — vector de pesos La tarea de regresión lineal en el lenguaje de las matrices puede formularse así:

$$y = Xw$$

El objetivo de entrenamiento es entonces encontrar esa $w$ w que minimice la distancia L2 (ECM) entre $Xw$ y $y$:

$$\min_w d_2(Xw, y) \quad \text{or} \quad \min_w \text{MSE}(Xw, y)$$

Parece que hay una solución analítica para lo anteriormente expuesto:

$$w = (X^T X)^{-1} X^T y$$

La fórmula anterior puede servir para encontrar los pesos $w$ y estos últimos pueden utilizarse para calcular los valores predichos

$$\hat{y} = X_{val} w$$

Divide todos los datos correspondientes a las etapas de entrenamiento/prueba respetando la proporción 70:30. Utiliza la métrica RECM para evaluar el modelo.

[29]:
```
class MyLinearRegression:

    def __init__(self):

        self.weights = None

    def fit(self, X, y):

        # añadir las unidades
        X2 = np.append(np.ones([len(X), 1]), X, axis=1)
        self.weights = np.linalg.inv(X2.T.dot(X2)).dot(X2.T).dot(y)

    def predict(self, X):

        # añadir las unidades
        X2 = np.append(np.ones([len(X), 1]), X, axis=1)
        y_pred = X2.dot(self.weights)

        return y_pred
```

[30]:
```
def eval_regressor(y_true, y_pred):

    rmse = math.sqrt(sklearn.metrics.mean_squared_error(y_true, y_pred))
    print(f'RMSE: {rmse:.2f}')

    r2_score = math.sqrt(sklearn.metrics.r2_score(y_true, y_pred))
    print(f'R2: {r2_score:.2f}')
```

[31]:
```
#Regresión lineal con datos *no* escalados
X = df[['age', 'gender', 'income', 'family_members']].to_numpy()
y = df['insurance_benefits'].to_numpy()

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=12345)

lr = MyLinearRegression()

lr.fit(X_train, y_train)
print(lr.weights)

y_test_pred = lr.predict(X_test)
eval_regressor(y_test, y_test_pred)

#print('-')
#print(y_test_pred)
```

```
[-9.43539012e-01  3.57495491e-02  1.64272726e-02 -2.60743659e-07
 -1.16902127e-02]
RMSE: 0.34
R2: 0.66
```

[32]:
```
#Regresión lineal con datos escalados
X = df_scaled[['age', 'gender', 'income', 'family_members']].to_numpy()
y = df_scaled['insurance_benefits'].to_numpy()

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=12345)

lr_scaled = MyLinearRegression()

lr_scaled.fit(X_train, y_train)
print(lr_scaled.weights)

y_test_pred = lr_scaled.predict(X_test)
eval_regressor(y_test, y_test_pred)

#print('-')
```

```
#print(y_test_pred)
```

```
[-0.94353901  2.32372069  0.01642727 -0.02059875 -0.07014128]
RMSE: 0.34
R2: 0.66
```

Podemos observar que el regresor lineal no cambia su desempeño con los datos escalados, esto se debe a que las métricas R^2 y RMSE no se ven influenciadas por la escalación de los datos, ambas métricas se calculan en la misma variable independiente (target).

El RMSE mide el promedio del error de predicción, por lo tanto, las distancias entre los datos escalados y las distancias entre los datos no escalados, será la misma.

Algo similar sucede con la métrica R^2, esta métrica está calculada a partir del RMSE y explica la varianza de la variable objetivo , por lo tanto, al igual que en la métrica RMSE, las distancias son las mismas, aún cuando se escalan los datos.

***Nota:***

Investigando un poco más a fondo encontramos que en modelos de regresión lineal como Ridge y Lasso si cobra total relevancia la escalación de los datos.

# 6 Tarea 4. Ofuscar datos

Lo mejor es ofuscar los datos multiplicando las características numéricas (recuerda que se pueden ver como la matriz $X$) por una matriz invertible $P$.

$$X' = X \times P$$

Trata de hacerlo y comprueba cómo quedarán los valores de las características después de la transformación. Por cierto, la propiedad de invertibilidad es importante aquí, así que asegúrate de que $P$ sea realmente invertible.

Puedes revisar la lección 'Matrices y operaciones matriciales -> Multiplicación de matrices' para recordar la regla de multiplicación de matrices y su implementación con NumPy.

[33]:
```
personal_info_column_list = ['gender', 'age', 'income', 'family_members']
df_pn = df[personal_info_column_list]
```

[34]:
```
X = df_pn.to_numpy()
X.shape
```

[34]:
```
(5000, 4)
```

Generar una matriz aleatoria $P$.

[35]:
```
rng = np.random.default_rng(seed=42)
P = rng.random(size=(X.shape[1], X.shape[1]))
P
```

[35]:
```
array([[0.77395605, 0.43887844, 0.85859792, 0.69736803],
       [0.09417735, 0.97562235, 0.7611397 , 0.78606431],
       [0.12811363, 0.45038594, 0.37079802, 0.92676499],
       [0.64386512, 0.82276161, 0.4434142 , 0.22723872]])
```

Comprobar que la matriz P sea invertible

[36]:
```
np.linalg.inv(P)
```

[36]:
```
array([[ 0.41467992, -1.43783972,  0.62798546,  1.14001268],
       [-1.06101789,  0.44219337,  0.1329549 ,  1.18425933],
       [ 1.42362442,  1.60461607, -2.0553823 , -1.53699695],
       [-0.11128575, -0.65813802,  1.74995517, -0.11816316]])
```

Mediante np.linalg.inv() comprobamos que la matriz P es invertible

¿Puedes adivinar la edad o los ingresos de los clientes después de la transformación?

[37]:
```
XP = X.dot(P)
XP
```

[37]:
```
array([[ 6359.71527314, 22380.40467609, 18424.09074184, 46000.69669016],
       [ 4873.29406479, 17160.36702982, 14125.78076133, 35253.45577301],
       [ 2693.11742928,  9486.397744  ,  7808.83156024, 19484.86063067],
       …,
       [ 4346.2234249 , 15289.24126492, 12586.16264392, 31433.50888552],
       [ 4194.09324155, 14751.9910242 , 12144.02930637, 30323.88763426],
       [ 5205.46827354, 18314.24814446, 15077.01370762, 37649.59295455]])
```

[38]:
```
obfuscated=pd.DataFrame(XP,columns=df_pn.columns)
obfuscated.head()
```

[38]:

| | gender | age | income | family_members |
|---|---|---|---|---|
| 0 | 6359.715273 | 22380.404676 | 18424.090742 | 46000.696690 |
| 1 | 4873.294065 | 17160.367030 | 14125.780761 | 35253.455773 |
| 2 | 2693.117429 | 9486.397744 | 7808.831560 | 19484.860631 |
| 3 | 5345.603937 | 18803.227203 | 15479.148373 | 38663.061863 |
| 4 | 3347.176735 | 11782.829283 | 9699.998942 | 24211.273378 |

**No se pueden adivinar los ingresos de los clientes después de la transformación, debido a que quedan totalmente ofuscados**

¿Puedes recuperar los datos originales de $X'$ si conoces $P$? Intenta comprobarlo a través de los cálculos moviendo $P$ del lado derecho de la fórmula anterior al izquierdo. En este caso las reglas de la multiplicación matricial son realmente útiles

**La formula que vamos a utilizar para recuperar los datos es:**

$$X = P^{-1} \times X'$$

**donde**

$$X' = X \times P$$

[39]:
```
recovered_array = XP.dot(np.linalg.inv(P))
recovered_array
```

[39]:
```
array([[ 1.00000000e+00,  4.10000000e+01,  4.96000000e+04,
         1.00000000e+00],
       [-4.47363596e-12,  4.60000000e+01,  3.80000000e+04,
         1.00000000e+00],
       [-2.51586878e-12,  2.90000000e+01,  2.10000000e+04,
         9.52452315e-13],
       …,
       [-1.92837871e-12,  2.00000000e+01,  3.39000000e+04,
         2.00000000e+00],
       [ 1.00000000e+00,  2.20000000e+01,  3.27000000e+04,
         3.00000000e+00],
       [ 1.00000000e+00,  2.80000000e+01,  4.06000000e+04,
         1.00000000e+00]])
```

[40]:
```
recovered_df = pd.DataFrame(recovered_array, columns = df_pn.columns)
recovered_df.head(5)
```

[40]:

| | gender | age | income | family_members |
|---|---|---|---|---|
| 0 | 1.000000e+00 | 41.0 | 49600.0 | 1.000000e+00 |
| 1 | -4.473636e-12 | 46.0 | 38000.0 | 1.000000e+00 |
| 2 | -2.515869e-12 | 29.0 | 21000.0 | 9.524523e-13 |
| 3 | -4.844982e-12 | 21.0 | 41700.0 | 2.000000e+00 |
| 4 | 1.000000e+00 | 28.0 | 26100.0 | -1.019907e-13 |

Muestra los tres casos para algunos clientes - Datos originales - El que está transformado - El que está invertido (recuperado)

[41]:
```
#Datos originales
df_pn.iloc[105:107]
```

[41]:

| | gender | age | income | family_members |
|---|---|---|---|---|
| 105 | 1 | 35 | 44300.0 | 0 |
| 106 | 1 | 45 | 41600.0 | 2 |

[42]:
```
#Datos ofuscados
obfuscated.iloc[105:107]
```

[42]:

| | gender | age | income | family_members |
|---|---|---|---|---|
| 105 | 5679.504091 | 19986.682710 | 16453.850961 | 41083.898625 |
| 106 | 5335.826786 | 18782.042424 | 15461.194521 | 38589.948275 |

[48]:
```
#Datos recuperados
recovered_df.iloc[105:107]
```

[48]:

| | gender | age | income | family_members |
|---|---|---|---|---|
| 105 | 1.0 | 35.0 | 44300.0 | -1.612182e-12 |
| 106 | 1.0 | 45.0 | 41600.0 | 2.000000e+00 |

Seguramente puedes ver que algunos valores no son exactamente iguales a los de los datos originales. ¿Cuál podría ser la razón de ello?

**El motivo por el cual los valores recuperados no son exactamente iguales es debido a la precisión numérica, las computadoras al realizar cálculos con valores flotantes, no pueden tomar infinitos decimales, por lo tanto se trunca el cálculo con una cantidad fija de decimales y eso puede causar una pérdida mínima de datos.**

## 6.1 Prueba de que la ofuscación de datos puede funcionar con regresión lineal

En este proyecto la tarea de regresión se ha resuelto con la regresión lineal. Tu siguiente tarea es demostrar *analytically* que el método de ofuscación no afectará a la regresión lineal en términos de valores predichos, es decir, que sus valores seguirán siendo los mismos. ¿Lo puedes creer? Pues no hace falta que lo creas, ¡tienes que que demostrarlo!

Entonces, los datos están ofuscados y ahora tenemos $X \times P$ en lugar de tener solo $X$. En consecuencia, hay otros pesos $w_P$ como

$$w = (X^T X)^{-1} X^T y \quad \Rightarrow \quad w_P = [(XP)^T XP]^{-1} (XP)^T y$$

¿Cómo se relacionarían $w$ y $w_P$ si simplificáramos la fórmula de $w_P$ anterior?

¿Cuáles serían los valores predichos con $w_P$?

¿Qué significa esto para la calidad de la regresión lineal si esta se mide mediante la RECM? Revisa el Apéndice B Propiedades de las matrices al final del cuaderno. ¡Allí encontrarás fórmulas muy útiles!

No es necesario escribir código en esta sección, basta con una explicación analítica.

**Respuesta**

Utilizando las propiedades de las matrices y realizando despejes:

Para los valores no ofuscados:

$$\hat{y} = X_{val} w$$

por lo tanto para valores ofuscados:

$$\hat{y} = X P_{val} w_P$$

**Prueba analítica**

$$\hat{y} = XP_{val} w_P$$

donde w_P =

$$w_P = [(XP)^T XP]^{-1} (XP)^T y$$

Por tanto:

$$\hat{y} = XP_{val} \cdot [(XP)^T XP]^{-1} (XP)^T y$$

utilizando propiedades de matrices: * Propiedad de identidad multiplicativa

$IA = AI = A$

$A^{-1}A = AA^{-1} = I$

$(AB)^{-1} = B^{-1}A^{-1}$

Entonces:

$$\hat{y} = XP_{\text{val}} \cdot XP^{-1} \cdot [(XP)^T)]^{-1} \cdot (XP)^T \cdot y$$

los términos

$$(XP)^T \quad y \quad \left[(XP)^T\right]^{-1}$$

Son iguales a esta propiedad: $A^{-1}A = AA^{-1} = I$

Por lo tanto son iguales a la Matriz identidad, lo que equivale al valor 1 escalar.

Sustituyendo:

$$\hat{y} = XP \cdot [XP]^{-1} \cdot y$$

Utilizando la misma propiedad anterior, los términos:

$$XP \cdot [XP]^{-1}$$

son equivalentes de igual manera a la matriz Identidad, que es lo mismo que un valor 1 escalar.

por último sustituyendo, nuestra fórmula queda de esta manera:

$$y = y$$

Con esto concluimos que al ofuscar los datos no afectamos la integridad y el funcionamiento del modelo.

## 6.2 Prueba de regresión lineal con ofuscación de datos

Ahora, probemos que la regresión lineal pueda funcionar, en términos computacionales, con la transformación de ofuscación elegida. Construye un procedimiento o una clase que ejecute la regresión lineal opcionalmente con la ofuscación. Puedes usar una implementación de regresión lineal de scikit-learn o tu propia implementación. Ejecuta la regresión lineal para los datos originales y los ofuscados, compara los valores predichos y los valores de las métricas RMSE y $R^2$. ¿Hay alguna diferencia?

**Procedimiento**

- Crea una matriz cuadrada $P$ de números aleatorios.
- Comprueba que sea invertible. Si no lo es, repite el primer paso hasta obtener una matriz invertible.
- <¡ tu comentario aquí !>
- Utiliza $XP$ como la nueva matriz de características

[44]:
```
# agregado
X = df[['age', 'gender', 'income', 'family_members']].to_numpy()
y = df['insurance_benefits'].to_numpy()


rng = np.random.default_rng(seed=100)
P = rng.random(size=(X.shape[1], X.shape[1]))
P
```

[44]:
```
array([[0.83498163, 0.59655403, 0.28886324, 0.04295157],
       [0.9736544 , 0.5964717 , 0.79026316, 0.91033938],
       [0.68815445, 0.18999147, 0.98147898, 0.28474005],
       [0.62927317, 0.58103648, 0.59991227, 0.53524811]])
```

[45]:
```
XP = X @ P
XP
```

[45]:
```
array([[34168.29777497,  9449.21330332, 48694.59100746, 14126.31308276],
       [26188.90743666,  7247.69851045, 37310.08888523, 10822.63292103],
       [14475.45786671,  4007.12100792, 20619.43562718,  5980.78664592],
       …,
       [23346.39395231,  6453.80410134, 33279.11453264,  9654.61722322],
       [22523.88150603,  6228.18494951, 32103.30765781,  9314.46065383],
       [27964.05298544,  7731.53484048, 39857.52495967, 11563.09426218]])
```

[46]:
```
XP_train, XP_test, y_train, y_test = train_test_split(XP, y, test_size=0.3,␣
 ↪random_state=12345)

new_lr = MyLinearRegression()

new_lr.fit(XP_train, y_train)
print(new_lr.weights)
```

```
yP_test_pred =new_lr.predict(XP_test)
eval_regressor(y_test, yP_test_pred)
```

```
[-9.43539228e-01  1.25918191e-01 -8.12192049e-02 -7.24009967e-02
 -5.63358602e-04]
RMSE: 0.34
R2: 0.66
```

[47]:
```
y_test_pred,yP_test_pred
```

[47]:
```
(array([ 0.17926625,  0.80931996,  0.45614337, …, -0.00870507,
         0.1068024 ,  0.44539183]),
 array([ 0.17926647,  0.80932011,  0.45614329, …, -0.00870513,
         0.10680265,  0.44539178]))
```

Observamos que los valores predichos en la regresión con los datos *no ofuscados* son prácticamente iguales a los valores predichos por la regresión con los datos *ofuscados*, se tiene una muy ligera diferencia entre los datos, por los motivos ya explicados respecto a la precisión de cálculo y los valores flotantes.

# 7 Conclusiones

En este proyecto pudimos resolver de manera exitosa las tareas que la compañía de seguros nos solicitó: * Encontramos clientes similares a un cliente determinado, esto con fines de marketing * Se encontró la probabilidad de que un nuevo cliente reciba una prestación (cualquiera) del seguro, se comprobó que el modelo predictivo K-N Vecinos funciona mejor que el modelo dummy con probabilidades fijas, siempre y cuando se entrene el modelo en condiciones óptimas con datos escalados y con la métrica de distancia adecuada para el problema.

- Se predijo el número de prestaciones de seguro que un nuevo cliente pueda recibir mediante un modelo de regresión lineal, arrojando métricas de desempeño RMSE = 0.36 y un R2 = 0.66,
- Se realizaron labores de ofuscación (enmascaramiento) de los datos para proteger los datos personales de los clientes, sin afectar al modelo, se realizó una comprobación analítica y práctica donde el modelo de regresión lineal nos arrojó las mismas métricas de desempeño RMSE = 0.36 y un R2 = 0.66

Dentro de las observaciones que el equipo de Ciencia de datos encontró:

- El modelo K Neighbors Classifier con las características no escaladas presentó un desempeño máximo con un valor K=9, donde el F1 Score fue de .6299, cabe mencionar que solamente 4 de 10 iteraciones de valores K fueron superiores al modelo dummy con F1 Score $> 0.2$
- El modelo K Neighbors Classifier presentó el mejor desempeño con las características escaladas y un valor K = 9, donde el F1 Score fue de 0.955
- Podemos entonces concluir que el comportamiento del modelo de clasificación por K-N Vecinos es óptimo cuando los datos se escalan, también resaltamos la utilidad de realizar iteraciones de los K vecinos para verificar el mejor comportamiento.

El comportamiento del modelo K-N Vecinos está altamente influenciado por la escala de las características, esto se debe a la sensibilidad de la escala, el modelo KNN calcula las distancias entre los puntos de datos para encontrar sus vecinos más cercanos, si tenemos una (o más) características con escalas muy distantes entre sí, las distancias de esta característica van a aportar más peso al resultado final que las restantes características.

En el modelo de regresión lineal podimos realizar el enmascaramiento de los datos mediante técnicas de ofuscación de datos, utilizamos herramientas de álgebra lineal para generar una codificación a partir de una matriz de datos generados de manera aleatoria.

En la ofuscación de los datos podemos tener pérdida de información mínima, esto debido a la precisión numérica, las computadoras al realizar cálculos con valores flotantes, no pueden tomar infinitos decimales, por lo tanto se trunca el cálculo con una cantidad fija de decimales y eso puede causar una pérdida mínima de datos.

La ofuscación de los datos puede ser una característica crucial para ciertos clientes que manejan información sensible, por ejemplo datos de salud de los usuarios, datos bancarios, fiscales, datos hipotecarios, etc.

# 8 Apéndices

## 8.1 Apéndice A: Escribir fórmulas en los cuadernos de Jupyter

Puedes escribir fórmulas en tu Jupyter Notebook utilizando un lenguaje de marcado proporcionado por un sistema de publicación de alta calidad llamado LaTeX (se pronuncia como "Lah-tech"). Las fórmulas se verán como las de los libros de texto.

Para incorporar una fórmula a un texto, pon el signo de dólar (\$) antes y después del texto de la fórmula, por ejemplo: $\frac{1}{2} \times \frac{3}{2} = \frac{3}{4}$ or $y = x^2, x \geq 1$.

Si una fórmula debe estar en el mismo párrafo, pon el doble signo de dólar (\
) antes y después del texto de la fórmula, por ejemplo:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i.$$

El lenguaje de marcado de LaTeX es muy popular entre las personas que utilizan fórmulas en sus artículos, libros y textos. Puede resultar complicado, pero sus fundamentos son sencillos. Consulta esta ficha de ayuda (materiales en inglés) de dos páginas para aprender a componer las fórmulas más comunes.

## 8.2 Apéndice B: Propiedades de las matrices

Las matrices tienen muchas propiedades en cuanto al álgebra lineal. Aquí se enumeran algunas de ellas que pueden ayudarte a la hora de realizar la prueba analítica de este proyecto.

Distributividad

$A(B + C) = AB + AC$

No conmutatividad

$AB \neq BA$

Propiedad asociativa de la multiplicación

$(AB)C = A(BC)$

Propiedad de identidad multiplicativa

$IA = AI = A$

$A^{-1}A = AA^{-1} = I$

$(AB)^{-1} = B^{-1}A^{-1}$

Reversibilidad de la transposición de un producto de matrices,

$(AB)^T = B^T A^T$