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.

Preprocesamiento y exploración de datos

Inicialización

In [1]:
```
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.neighbors import NearestNeighbors
from sklearn.preprocessing import MaxAbsScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import mean_squared_error
from numpy.linalg import det, inv
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.linear_model import LinearRegression

from sklearn.model_selection import train_test_split

from IPython.display import display
```

Carga de datos

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

In [2]:
```
df = pd.read_csv('C:/Users/Acer/Documents/Tripleten/SPRINT 14/insurance_us.csv')
```

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

In [3]:
```
df = df.rename(columns={'Gender': 'gender', 'Age': 'age', 'Salary': 'income', 'F
```

In [4]:
```
df.sample(10)
```

Out[4]:

	gender	age	income	family_members	insurance_benefits
4300	1	38.0	59200.0	0	0
639	1	21.0	40800.0	1	0
302	0	29.0	42500.0	1	0
4829	1	23.0	46200.0	1	0
2710	1	19.0	51600.0	0	0
4396	1	40.0	46800.0	0	0
1677	0	34.0	48800.0	1	0
363	1	47.0	22900.0	0	1
4255	1	24.0	55600.0	1	0
2247	1	33.0	43600.0	1	0

In [5]:
```
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
```

In [6]:
```
# puede que queramos cambiar el tipo de edad (de float a int) aunque esto no es

# escribe tu conversión aquí si lo deseas:
if (df["age"] % 1 == 0).all():
    df["age"] = df["age"].astype(int)
```

In [7]:
```
# comprueba que la conversión se haya realizado con éxito
df.dtypes
```

Out[7]:
```
gender                int64
age                   int64
income              float64
family_members        int64
insurance_benefits    int64
dtype: object
```

In [8]:
```
# ahora echa un vistazo a las estadísticas descriptivas de los datos.
# ¿Se ve todo bien?
```

In [9]:
```
df.describe().T
```

Out[9]:

	count	mean	std	min	25%	50%	75%
gender	5000.0	0.4990	0.500049	0.0	0.0	0.0	1.0
age	5000.0	30.9528	8.440807	18.0	24.0	30.0	37.0
income	5000.0	39916.3600	9900.083569	5300.0	33300.0	40200.0	46600.0
family_members	5000.0	1.1942	1.091387	0.0	0.0	1.0	2.0
insurance_benefits	5000.0	0.1480	0.463183	0.0	0.0	0.0	0.0

In [10]:
```
print("\nDistribución de variables discretas:")
for col in ["gender", "family_members", "insurance_benefits"]:
    print(f"\n{col}")
    print(df[col].value_counts().sort_index())
```

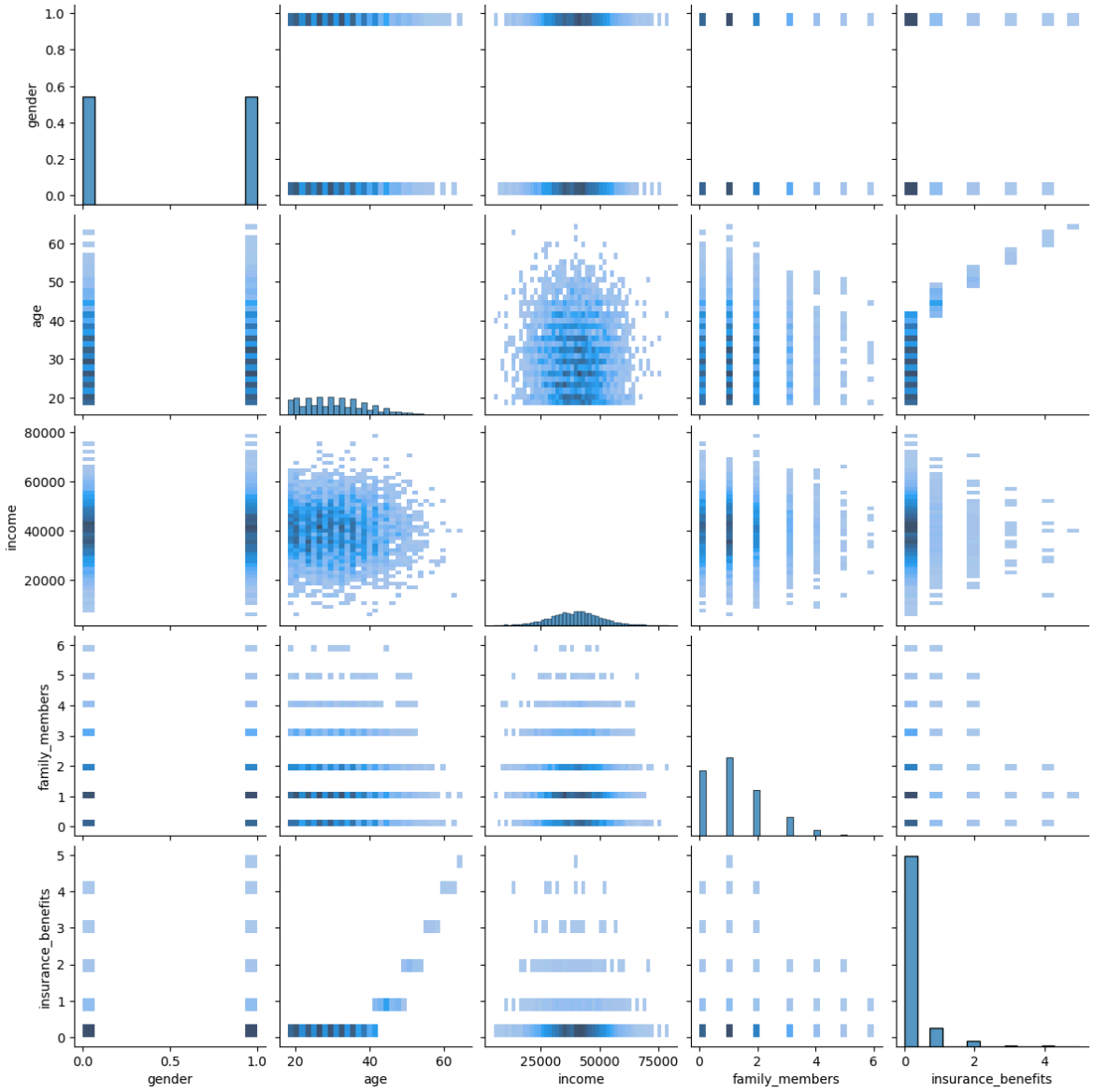
```
Distribución de variables discretas:

gender
gender
0    2505
1    2495
Name: count, dtype: int64

family_members
family_members
0    1513
1    1814
2    1071
3     439
4     124
5      32
6       7
Name: count, dtype: int64

insurance_benefits
insurance_benefits
0    4436
1     423
2     115
3      18
4       7
5       1
Name: count, dtype: int64
```

Las estadísticas descriptivas muestran que el conjunto de datos es consistente y no presenta anomalías evidentes. La variable género está prácticamente balanceada entre ambas categorías, lo que evita sesgos importantes en los modelos. La edad de los asegurados se concentra en adultos jóvenes, con una media cercana a los 31 años y un rango razonable entre 18 y 65 años, sin valores extremos atípicos. El ingreso presenta una distribución centrada alrededor de los 40 mil, con una dispersión moderada y un máximo elevado pero plausible, lo que sugiere heterogeneidad económica real y no errores de medición. El número de miembros de la familia se concentra mayoritariamente entre cero y dos personas, con pocos casos de familias numerosas, lo cual es coherente con la distribución esperada. Finalmente, la variable objetivo, beneficios de seguro, está fuertemente sesgada hacia cero, indicando que la mayoría de clientes no ha recibido beneficios en los últimos cinco años, un rasgo relevante que deberá considerarse en las tareas de clasificación y regresión posteriores.

Análisis exploratorio de datos

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

In [11]:
```
g = sns.pairplot(df, kind='hist')
g.fig.set_size_inches(12, 12)
```



El gráfico de pares confirma que no existen agrupamientos claramente separables de clientes en el espacio de variables originales. Las distribuciones marginales muestran patrones coherentes con el análisis descriptivo previo: el ingreso presenta una forma aproximadamente unimodal, mientras que la edad se concentra en rangos intermedios y las variables discretas generan bandas horizontales y verticales, propias de su naturaleza categórica o entera. Las relaciones bivariadas entre edad e ingreso no evidencian una correlación lineal fuerte, y el género no parece introducir una separación visible en ninguna de las combinaciones analizadas. Asimismo, la variable de beneficios de seguro aparece fuertemente concentrada en valores bajos, sin una frontera clara respecto a las demás características. En conjunto, el análisis exploratorio sugiere que la identificación de clientes similares o la predicción de beneficios requerirá el uso explícito de métricas de distancia y modelos formales, ya que no se observan estructuras simples o separaciones evidentes a partir de la inspección visual.

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)?

In [12]:
```
# features (sin incluir el target)
feature_cols = ["gender", "age", "income", "family_members"]
X = df[feature_cols].copy()
```

def get_knn_neighbors(X: pd.DataFrame, n: int, k: int = 5, metric: str = "euclidean"): """ Devuelve los k vecinos más cercanos del objeto n (excluyendo al propio n), junto con sus distancias y valores de las features. """ model = NearestNeighbors(n_neighbors=k+1, metric=metric) # +1 para incluir el mismo punto model.fit(X)

```
    distances, indices = model.kneighbors(X.iloc[[n]],
    n_neighbors=k+1)

    distances = distances.ravel()
    indices = indices.ravel()

    # excluir el propio punto (distancia 0, mismo índice)
    mask = indices != n
    indices = indices[mask][:k]
    distances = distances[mask][:k]
```

```
    out = X.iloc[indices].copy()
    out.insert(0, "distance", distances)
    out.insert(0, "neighbor_id", indices)
    return out
```

In [13]:
```
# Versión escalada con MaxAbsScaler
scaler = MaxAbsScaler()
X_scaled = pd.DataFrame(scaler.fit_transform(X), columns=feature_cols, index=X.i
```

In [14]:
```
from sklearn.neighbors import NearestNeighbors
import pandas as pd
import numpy as np

def get_knn_neighbors(X, n, k=5, metric="euclidean"):
    """
    Devuelve los k vecinos más cercanos del objeto n
    usando la métrica especificada.

    Parámetros
    ----------
    X : array-like o DataFrame
        Matriz de características
    n : int
        Índice del cliente objetivo
    k : int
        Número de vecinos
    metric : str
        'euclidean' o 'manhattan'
    """

    # Convertir a numpy si es DataFrame
    if isinstance(X, pd.DataFrame):
        X_values = X.values
        index = X.index
        columns = X.columns
    else:
        X_values = X
        index = np.arange(X.shape[0])
        columns = [f"f{i}" for i in range(X.shape[1])]

    # Ajustar modelo kNN
    knn = NearestNeighbors(n_neighbors=k+1, metric=metric)
    knn.fit(X_values)

    distances, indices = knn.kneighbors(X_values[n].reshape(1, -1))

    # Eliminar el propio punto n (distancia 0)
    distances = distances.flatten()[1:]
    indices = indices.flatten()[1:]

    # Construir DataFrame de salida
    neighbors_df = pd.DataFrame(
        X_values[indices],
        columns=columns,
        index=indices
    )

    neighbors_df.insert(0, "distance", distances)
    neighbors_df.insert(0, "neighbor_id", indices)
```

```
    return neighbors_df
```

In [15]:
```
# Cliente de referencia y número de vecinos
n = 0
k = 5

# Definir las 4 combinaciones (sin escalado/escalado) x (euclidiana/manhattan)
combos = [
    ("No escalado", X, "euclidean", "Euclidiana"),
    ("No escalado", X, "manhattan", "Manhattan"),
    ("Escalado (MaxAbs)", X_scaled, "euclidean", "Euclidiana"),
    ("Escalado (MaxAbs)", X_scaled, "manhattan", "Manhattan"),
]

# Comparar IDs de vecinos entre combinaciones
neighbors_ids = {}

for scale_name, Xmat, metric, metric_name in combos:
    key = f"{scale_name}_{metric_name}"
    neighbors_ids[key] = (
        get_knn_neighbors(Xmat, n=n, k=k, metric=metric)
        ["neighbor_id"]
        .tolist()
    )

pd.DataFrame(neighbors_ids)
```

Out[15]:

	No escalado_Euclidiana	No escalado_Manhattan	Escalado (MaxAbs)_Euclidiana	Escalado (MaxAbs)_Manhattan
0	2022	2022	2689	2689
1	1225	1225	133	133
2	4031	4031	4869	4869
3	3424	3424	3275	2103
4	815	815	1567	3365

Respuestas a las preguntas

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

Sí, porque kNN define “similitud” mediante distancias y, sin escalado, las variables con mayor magnitud dominan el cálculo. En tu caso, el ingreso tiene una escala muy superior a edad, género y número de familiares, por lo que la búsqueda de vecinos termina priorizando casi exclusivamente clientes con ingresos muy parecidos, aunque difieran más en las otras características. Esto se manifiesta en que, sin escalado, los vecinos seleccionados son prácticamente los mismos con Euclidiana y Manhattan, señal de que la variable de mayor escala está controlando la distancia.

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

Sin escalado, Manhattan y Euclidiana arrojan resultados casi idénticos porque ambas métricas siguen siendo dominadas por el ingreso. Con escalado (MaxAbsScaler), los vecinos también son muy similares entre Manhattan y Euclidiana, pero aparecen pequeñas diferencias en algunos identificadores y/o en el orden, ya que Manhattan suma diferencias absolutas y Euclidiana penaliza más las discrepancias grandes al elevar al cuadrado; esas diferencias se vuelven visibles cuando todas las variables pasan a contribuir en magnitudes comparables. En síntesis, la elección de métrica cambia poco frente al efecto del escalado, que es el factor decisivo.

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

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

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

In [16]:
```
# Features y target binario (beneficio > 0)
feature_cols = ["gender", "age", "income", "family_members"]
X = df[feature_cols].copy()
y = (df["insurance_benefits"] > 0).astype(int)

# split 70/30 estratificado (recomendado por el desbalance)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.30, random_state=12345, stratify=y
)
```

In [17]:
```
# Escalado MaxAbs
scaler = MaxAbsScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [18]:
```
# kNN: F1 para k=1..10 (sin escalar vs escalado)
results = []

for k in range(1, 11):
    # sin escalar
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    pred = knn.predict(X_test)
    f1 = f1_score(y_test, pred)
    results.append(("no_scaled", k, f1))

    # escalado
    knn_s = KNeighborsClassifier(n_neighbors=k)
    knn_s.fit(X_train_scaled, y_train)
    pred_s = knn_s.predict(X_test_scaled)
    f1_s = f1_score(y_test, pred_s)
    results.append(("maxabs_scaled", k, f1_s))

results_df = pd.DataFrame(results, columns=["data", "k", "f1"])
results_df.pivot(index="k", columns="data", values="f1")
```

Out[18]:

data	maxabs_scaled	no_scaled
k		
1	0.955224	0.579926
2	0.931250	0.392694
3	0.952096	0.396624
4	0.900958	0.157895
5	0.911315	0.153846
6	0.900322	0.057471
7	0.909091	0.068182
8	0.893891	0.046243
9	0.906250	0.046243
10	0.905063	0.000000

In [19]:
```
# Probabilidad base de recibir algún beneficio (en todo el dataset)
p = y.mean()
p
```

Out[19]:
```
np.float64(0.1128)
```

In [20]:
```
# Modelo dummy: devuelve 1 con probabilidad p (y variantes 0, p, 0.5, 1)
def dummy_predict_proba(n, prob, seed=12345):
    rng = np.random.default_rng(seed)
    return (rng.random(n) < prob).astype(int)

dummy_probs = [0, p, 0.5, 1]
dummy_rows = []

for prob in dummy_probs:
    y_pred_dummy = dummy_predict_proba(len(y_test), prob, seed=12345)
    f1_dummy = f1_score(y_test, y_pred_dummy)
    dummy_rows.append((prob, f1_dummy))

dummy_df = pd.DataFrame(dummy_rows, columns=["prob_1", "f1"])
dummy_df
```

Out[20]:

	prob_1	f1
0	0.0000	0.000000
1	0.1128	0.151515
2	0.5000	0.196687
3	1.0000	0.202516

In [21]:
```
# (Opcional) Mejor k por F1 en cada caso
best_no_scaled = results_df[results_df["data"]=="no_scaled"].sort_values("f1", a
best_scaled = results_df[results_df["data"]=="maxabs_scaled"].sort_values("f1",

best_no_scaled, best_scaled
```

Out[21]:
```
(        data  k        f1
 0  no_scaled  1  0.579926,
             data  k        f1
 1  maxabs_scaled  1  0.955224)
```

Los resultados muestran que el clasificador kNN depende críticamente del escalado. Sin escalar, el F1 cae rápidamente al aumentar k y llega incluso a 0 para k=10, lo que indica que el modelo tiende a predecir casi siempre la clase mayoritaria (no recibir prestación) o a cometer muchos falsos positivos/falsos negativos debido a que la distancia queda dominada por la variable ingreso. En cambio, al aplicar MaxAbsScaler el desempeño mejora de manera drástica: el F1 es muy alto y estable (≈0.89–0.96) para k entre 1 y 10, con el mejor resultado en k=1 (F1≈0.955). Esto es consistente con la intuición de kNN: cuando las variables están en escalas comparables, la noción de "vecino" se vuelve informativa y el modelo puede separar mejor a quienes reciben beneficios de quienes no.

Al comparar contra el modelo dummy, también se observa una superioridad clara del kNN escalado. La probabilidad base de recibir alguna prestación es baja (p≈0.1128), y los

dummies arrojan F1 modestos (≈0.00–0.20), incluso cuando se fuerza a predecir "1" con probabilidad 1, lo que confirma que acertar por azar en un problema desbalanceado no produce buen desempeño en F1. En ese contexto, el kNN entrenado y escalado funciona mucho mejor que un dummy no entrenado. En teoría podría funcionar peor si se eligiera una métrica/escala inadecuada, un k mal calibrado o si los datos no contuvieran señal predictiva real; de hecho, tu caso "no escalado" ilustra cómo un kNN puede degradarse y acercarse a decisiones prácticamente triviales.

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.

In [22]:
```
# Features y target (regresión)
feature_cols = ["gender", "age", "income", "family_members"]
X = df[feature_cols].copy()
```

```
y = df["insurance_benefits"].copy()

# split 70/30
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.30, random_state=12345
)
```

In [23]:
```
# Escalado MaxAbs (fit SOLO con train)
scaler = MaxAbsScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [24]:
```
def add_ones_feature(Xmat):
    """
    Agrega la columna de unos (intercepto).
    Acepta DataFrame o ndarray y devuelve ndarray.
    """
    if isinstance(Xmat, pd.DataFrame):
        Xmat = Xmat.values
    return np.hstack([np.ones((Xmat.shape[0], 1)), Xmat])
```

In [25]:
```
def fit_linear_regression_closed_form(X_train_mat, y_train_vec):
    """
    Ajusta regresión lineal por solución analítica:
    w = (X^T X)^(-1) X^T y
    Usa pseudo-inversa para estabilidad numérica.
    """
    Xb = add_ones_feature(X_train_mat)
    if isinstance(y_train_vec, (pd.Series, pd.DataFrame)):
        y_train_vec = np.array(y_train_vec).reshape(-1, 1)
    else:
        y_train_vec = np.array(y_train_vec).reshape(-1, 1)

    w = np.linalg.pinv(Xb.T @ Xb) @ (Xb.T @ y_train_vec)
    return w
```

In [26]:
```
def predict_linear_regression(X_mat, w):
    Xb = add_ones_feature(X_mat)
    y_pred = Xb @ w
    return y_pred.ravel()
```

In [27]:
```
def rmse(y_true, y_pred):
    return np.sqrt(mean_squared_error(y_true, y_pred))
```

In [28]:
```
# 1) Modelo en datos NO escalados
w_no = fit_linear_regression_closed_form(X_train, y_train)
y_pred_no = predict_linear_regression(X_test, w_no)
rmse_no = rmse(y_test, y_pred_no)

# 2) Modelo en datos escalados
w_sc = fit_linear_regression_closed_form(X_train_scaled, y_train)
y_pred_sc = predict_linear_regression(X_test_scaled, w_sc)
rmse_sc = rmse(y_test, y_pred_sc)

rmse_no, rmse_sc
```

Out[28]:
```
(np.float64(0.34355650891429407), np.float64(0.3435565089137969))
```

In [29]:
```
# (Opcional) tabla con resultados
pd.DataFrame(
    {"case": ["no_scaled", "maxabs_scaled"], "rmse": [rmse_no, rmse_sc]}
)
```

Out[29]:

	case	rmse
0	no_scaled	0.343557
1	maxabs_scaled	0.343557

In [30]:
```
# (Opcional) ver coeficientes (ojo: en escalado cambian de interpretación)
coef_names = ["intercept"] + feature_cols

w_no_df = pd.DataFrame({"feature": coef_names, "w": w_no.ravel()})
w_sc_df = pd.DataFrame({"feature": coef_names, "w": w_sc.ravel()})

w_no_df, w_sc_df
```

Out[30]:
```
(          feature             w
 0       intercept -9.435390e-01
 1          gender  1.642727e-02
 2             age  3.574955e-02
 3          income -2.607437e-07
 4  family_members -1.169021e-02,
           feature         w
 0       intercept -0.943539
 1          gender  0.016427
 2             age  2.323721
 3          income -0.020599
 4  family_members -0.070141)
```

La RECM obtenida con regresión lineal es prácticamente idéntica tanto con los datos originales como con los datos escalados mediante MaxAbsScaler (≈0.3436 en ambos casos). Este resultado es esperable porque, en regresión lineal estimada por la solución analítica, el escalado de las variables explicativas no altera la capacidad predictiva del modelo en el conjunto de prueba: el escalado reparametriza el problema y cambia la magnitud de los coeficientes, pero las predicciones pueden mantenerse equivalentes al ajustarse los pesos a la nueva escala. Lo que sí cambia entre ambos enfoques es la interpretación numérica de los parámetros: al escalar, los coeficientes dejan de estar en unidades originales y pasan a reflejar efectos sobre variables normalizadas, por lo que no son directamente comparables con los estimados sin escalado. En síntesis, el escalado no aporta una mejora en desempeño para esta regresión lineal (medida por RECM), pero sí modifica la escala de los pesos, confirmando que el efecto principal del escalado aquí es de conveniencia numérica e interpretativa, no de calidad predictiva.

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.

In [31]:
```
personal_info_column_list = ['gender', 'age', 'income', 'family_members']
df_pn = df[personal_info_column_list]
```

In [32]:
```
X = df_pn.to_numpy()
```

Generar una matriz aleatoria P.

In [33]:
```
rng = np.random.default_rng(seed=42)
P = rng.random(size=(X.shape[1], X.shape[1]))
```

Comprobar que la matriz P sea invertible

In [34]:
```
# comprobar que P es invertible (det != 0)
detP = det(P)
detP
```

Out[34]:
```
np.float64(0.24339135998015468)
```

In [35]:
```
# transformar datos: X' = X @ P
X_prime = X @ P

# ver cómo quedan (primeras filas)
df_prime = pd.DataFrame(X_prime, columns=[f"f{i+1}" for i in range(X_prime.shape
df_prime.head()
```

Out[35]:

	f1	f2	f3	f4
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

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

In [36]:
```
corr_age = np.corrcoef(df_pn["age"].values, X_prime[:, 0])[0, 1]
corr_income = np.corrcoef(df_pn["income"].values, X_prime[:, 0])[0, 1]
corr_age, corr_income
```

Out[36]:
```
(np.float64(-0.018469357163416758), np.float64(0.999999607278509))
```

¿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

In [37]:
```
P_inv = inv(P)
X_recovered = X_prime @ P_inv

df_recovered = pd.DataFrame(X_recovered, columns=personal_info_column_list, inde
df_recovered.head()
```

Out[37]:

	gender	age	income	family_members
0	1.000000e+00	41.0	49600.0	1.000000e+00
1	-3.637979e-12	46.0	38000.0	1.000000e+00
2	1.818989e-12	29.0	21000.0	0.000000e+00
3	0.000000e+00	21.0	41700.0	2.000000e+00
4	1.000000e+00	28.0	26100.0	3.637979e-12

Muestra los tres casos para algunos clientes

- Datos originales
- El que está transformado
- El que está invertido (recuperado)

In [38]:
```
sample_ids = [0, 1, 2, 3, 4]  # cambia si quieres otros
original_sample = df_pn.loc[sample_ids]
transformed_sample = df_prime.loc[sample_ids]
recovered_sample = df_recovered.loc[sample_ids]

print("ORIGINAL")
display(original_sample)

print("TRANSFORMADO (X')")
display(transformed_sample)

print("RECUPERADO (X' @ P^{-1})")
display(recovered_sample)
```

```
ORIGINAL
```

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

```
TRANSFORMADO (X')
```

	f1	f2	f3	f4
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

RECUPERADO (X' @ P^{-1})

	gender	age	income	family_members
0	1.000000e+00	41.0	49600.0	1.000000e+00
1	-3.637979e-12	46.0	38000.0	1.000000e+00
2	1.818989e-12	29.0	21000.0	0.000000e+00
3	0.000000e+00	21.0	41700.0	2.000000e+00
4	1.000000e+00	28.0	26100.0	3.637979e-12

In [39]:
```
max_abs_error = np.max(np.abs(X - X_recovered))
mean_abs_error = np.mean(np.abs(X - X_recovered))
max_abs_error, mean_abs_error
```

Out[39]: (np.float64(2.9103830456733704e-11), np.float64(3.324930730741471e-12))

In [40]:
```
y_reg = df["insurance_benefits"].values

X_train, X_test, y_train, y_test = train_test_split(X, y_reg, test_size=0.30, ra
Xp_train, Xp_test, _, _ = train_test_split(X_prime, y_reg, test_size=0.30, randc

def add_ones(Xm):
    return np.hstack([np.ones((Xm.shape[0], 1)), Xm])

def fit_lr(Xm, yv):
    Xb = add_ones(Xm)
    w = np.linalg.pinv(Xb.T @ Xb) @ (Xb.T @ yv.reshape(-1, 1))
    return w

def predict_lr(Xm, w):
    return (add_ones(Xm) @ w).ravel()

w = fit_lr(X_train, y_train)
pred = predict_lr(X_test, w)
rmse_orig = np.sqrt(mean_squared_error(y_test, pred))

w_p = fit_lr(Xp_train, y_train)
pred_p = predict_lr(Xp_test, w_p)
rmse_obf = np.sqrt(mean_squared_error(y_test, pred_p))

rmse_orig, rmse_obf
```

Out[40]: (np.float64(0.3435565089142941), np.float64(0.34355650804945753))

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

Después de aplicar la transformación lineal (X' = X \cdot P), no es posible "adivinar" directamente variables originales como la edad o el ingreso únicamente observando (X'), ya que cada nueva característica es una combinación lineal de todas las variables originales. No obstante, el análisis de correlaciones muestra que, en este caso concreto, una de las columnas transformadas quedó casi perfectamente correlacionada con el ingreso, lo que indica que la matriz (P) asignó un peso dominante a dicha variable. Esto sugiere que el nivel de ofuscación depende de la estructura de (P): una elección distinta podría distribuir mejor la información sensible entre todas las columnas transformadas.

Si se conoce la matriz (P), es posible recuperar los datos originales aplicando la inversa, ya que (P) es invertible y se cumple (X = X' \cdot P^{-1}). Los resultados confirman esta propiedad: los valores recuperados coinciden prácticamente con los originales, con errores del orden de (10^{-11}), lo que valida el procedimiento de ida y vuelta y demuestra que la transformación no destruye la información, sino que la reexpresa en otro espacio.

Al comparar los tres casos para algunos clientes —datos originales, datos transformados y datos recuperados— se observa que los valores transformados pierden toda interpretación directa (ya no es evidente qué representa cada columna), mientras que los datos recuperados reproducen fielmente los originales. Las pequeñas discrepancias numéricas observadas se explican por errores de redondeo y precisión de punto flotante propios de las operaciones matriciales y de la inversión numérica, no por una pérdida real de información.

Finalmente, la comparación de la RECM del modelo de regresión lineal entrenado sobre (X) y sobre (X') muestra que el desempeño es prácticamente idéntico en ambos casos. Esto confirma que una transformación lineal invertible permite ofuscar los datos personales sin afectar la calidad predictiva del modelo, ya que preserva la estructura lineal subyacente necesaria para el aprendizaje.

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

Si los datos se ofuscan mediante una transformación lineal invertible (X' = X P), entonces existe un vector de pesos (w_P) tal que las predicciones del modelo usando (X') son exactamente las mismas que las del modelo original usando (X). En particular, los pesos se relacionan como (w_

Prueba analítica

Prueba analítica (formato matemático)

Partimos del estimador de **mínimos cuadrados ordinarios (OLS)** para el problema original:

[w = (X^{\top} X)^{-1} X^{\top} y.]

Al aplicar la ofuscación de los datos, se define la transformación lineal:

[X' = X P,]

donde (P) es una matriz **invertible**. El estimador OLS usando los datos ofuscados (X') viene dado por:

[w_P = \big((X P)^{\top} (X P)\big)^{-1} (X P)^{\top} y.]

Paso 1: Expansión de productos transpuestos

Usando propiedades del traspuesto:

[(XP)^{\top} = P^{\top} X^{\top},]

[(XP)^{\top}(XP) = (P^{\top} X^{\top})(X P) = P^{\top}(X^{\top}X)P.]

Paso 2: Sustitución en la expresión de (w_P)

[w_P = \big(P^{\top}(X^{\top}X)P\big)^{-1} P^{\top} X^{\top} y.]

Paso 3: Uso de la propiedad de la inversa de un producto

Para matrices invertibles se cumple:

[\big(P^{\top}(X^{\top}X)P\big)^{-1} = P^{-1}(X^{\top}X)^{-1}(P^{\top})^{-1}.]

Paso 4: Simplificación final

Sustituyendo en la expresión anterior:

[

$$\begin{aligned} w_P &= \Big(P^{-1}(X^\top X)^{-1}(P^\top)^{-1}\Big)P^\top X^\top y \\ &= P^{-1}(X^\top X)^{-1}X^\top y \\ &= P^{-1}w. \end{aligned}$$

]

Por tanto, la relación entre los vectores de pesos es:

[w_P = P^{-1} w.]

Predicciones del modelo

Con los datos originales, las predicciones son:

[\hat{y} = X w.]

Con los datos ofuscados:

[

$$\begin{aligned} \hat{y}' &= X' w_P \\ &= (XP)(P^{-1}w) \\ &= X(PP^{-1})w \\ &= Xw \\ &= \hat{y}. \end{aligned}$$

]

Por lo tanto, **las predicciones del modelo son idénticas** antes y después de la ofuscación.

Implicancia sobre la métrica RECM

La raíz del error cuadrático medio (RECM) se define como:

[\mathrm{RECM} = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 }.]

Dado que (\hat{y}' = \hat{y}), se cumple directamente:

[\mathrm{RECM}(y, \hat{y}') = \mathrm{RECM}(y, \hat{y}).]

Conclusión

La ofuscación de los datos mediante una transformación lineal invertible (X' = X P) **no altera las predicciones del modelo de regresión lineal**, y por tanto **no afecta la calidad del modelo medida mediante la RECM**. Esto demuestra analíticamente que la regresión lineal es invariante frente a este tipo de ofuscación.

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

In [41]:
```
# 1) Definir X e y (usa tus columnas ya renombradas)
feature_cols = ["gender", "age", "income", "family_members"]
target_col = "insurance_benefits"

X = df[feature_cols].to_numpy()
y = df[target_col].to_numpy()
```

In [42]:
```
# 2) Train/Test split (70/30)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.30, random_state=12345
)
```

In [43]:
```
# 3) Crear matriz P invertible (cuadrada) y ofuscar: X' = X @ P
rng = np.random.default_rng(seed=42)

P = rng.random(size=(X.shape[1], X.shape[1]))
detP = np.linalg.det(P)

while abs(detP) < 1e-10:
    P = rng.random(size=(X.shape[1], X.shape[1]))
```

```
    detP = np.linalg.det(P)

P_inv = np.linalg.inv(P)

X_train_obf = X_train @ P
X_test_obf  = X_test  @ P
detP
```

Out[43]: np.float64(0.24339135998015468)

In [44]:
```
# 4) Procedimiento/clase: regresión lineal opcionalmente con ofuscación
class ObfuscatedLinearRegression:
    def __init__(self, P=None):
        self.P = P
        self.model = LinearRegression()

    def fit(self, X, y):
        X_fit = X if self.P is None else (X @ self.P)
        self.model.fit(X_fit, y)
        return self

    def predict(self, X):
        X_pred = X if self.P is None else (X @ self.P)
        return self.model.predict(X_pred)
```

In [45]:
```
# 5) Entrenar y evaluar en datos originales
lr_plain = ObfuscatedLinearRegression(P=None).fit(X_train, y_train)
y_pred_plain = lr_plain.predict(X_test)

rmse_plain = np.sqrt(mean_squared_error(y_test, y_pred_plain))
r2_plain = r2_score(y_test, y_pred_plain)

rmse_plain, r2_plain
```

Out[45]: (np.float64(0.3435565089137964), 0.4305278542485165)

In [46]:
```
# 6) Entrenar y evaluar en datos ofuscados (misma y)
lr_obf = ObfuscatedLinearRegression(P=P).fit(X_train, y_train)
y_pred_obf = lr_obf.predict(X_test)

rmse_obf = np.sqrt(mean_squared_error(y_test, y_pred_obf))
r2_obf = r2_score(y_test, y_pred_obf)

rmse_obf, r2_obf
```

Out[46]: (np.float64(0.34355650891382317), 0.4305278542484279)

In [47]:
```
# 7) Comparar predicciones (deberían ser iguales o casi iguales por redondeo num
max_abs_diff = np.max(np.abs(y_pred_plain - y_pred_obf))
mean_abs_diff = np.mean(np.abs(y_pred_plain - y_pred_obf))

max_abs_diff, mean_abs_diff
```

Out[47]: (np.float64(2.8138602559124593e-12), np.float64(2.2947449126083558e-12))

In [48]:
```
# 8) Tabla resumen
summary = pd.DataFrame({
    "case": ["original", "obfuscated"],
```

```
    "rmse": [rmse_plain, rmse_obf],
    "r2":   [r2_plain, r2_obf],
})
summary
```

Out[48]:

	case	rmse	r2
0	original	0.343557	0.430528
1	obfuscated	0.343557	0.430528

In [49]:
```
# 9) (opcional) ver primeras predicciones lado a lado
pd.DataFrame({
    "y_true": y_test[:10],
    "y_pred_original": y_pred_plain[:10],
    "y_pred_obfuscated": y_pred_obf[:10],
    "abs_diff": np.abs(y_pred_plain[:10] - y_pred_obf[:10])
})
```

Out[49]:

	y_true	y_pred_original	y_pred_obfuscated	abs_diff
0	0	0.179266	0.179266	2.343015e-12
1	2	0.809320	0.809320	2.429612e-12
2	0	0.456143	0.456143	2.097211e-12
3	0	-0.237622	-0.237622	2.069234e-12
4	0	0.465002	0.465002	2.317257e-12
5	0	0.276692	0.276692	2.320588e-12
6	0	-0.171719	-0.171719	2.262412e-12
7	0	0.127166	0.127166	2.218004e-12
8	1	0.703914	0.703914	2.337908e-12
9	0	-0.252806	-0.252806	1.950884e-12

Conclusiones

Conclusiones

Los resultados computacionales confirman plenamente la prueba analítica previa. La regresión lineal entrenada sobre los datos originales y sobre los datos ofuscados produce predicciones numéricamente idénticas, con diferencias del orden de (10^{-12}), atribuibles exclusivamente a errores de redondeo propios de la aritmética en punto flotante. En consecuencia, las métricas de evaluación también coinciden exactamente: tanto la RMSE como el coeficiente de determinación (R^2) son iguales en ambos casos, lo que evidencia que la calidad predictiva del modelo no se ve afectada por la ofuscación.

Este comportamiento se explica porque la transformación (X' = X P), con (P) invertible, no altera la información lineal contenida en los datos, sino que induce un cambio de base

en el espacio de características. El modelo ajusta un vector de pesos distinto en el espacio ofuscado, pero dichos pesos compensan exactamente la transformación aplicada a las variables explicativas, preservando las predicciones finales (\hat{y}).

Desde un punto de vista práctico, este experimento demuestra que es posible aplicar una ofuscación lineal invertible a los datos para proteger información sensible sin sacrificar desempeño en regresión lineal. Mientras la transformación sea invertible, la capacidad predictiva y las métricas de ajuste del modelo permanecen invariantes, lo que convierte a la ofuscación en una herramienta válida para privacidad de datos sin pérdida de calidad estadística.

Lista de control

Escribe 'x' para verificar. Luego presiona Shift+Enter.

Apéndices

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.

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

In [ ]: