

ChatGPT - IA proyectos



divisiones-de-entrenamiento-prueba-ibm-machine-l.ipynb
File

Gracias por subir el archivo **divisiones-de-entrenamiento-prueba-ibm-machine-l.ipynb**. Para poder cumplir con tu pedido de explicar cada bloque de código de forma detallada, incluyendo teoría y fundamentos, procederé a abrir y analizar el notebook directamente. Un momento mientras lo examino...

Voy a comenzar con la **explicación detallada** de las primeras **tres celdas de código** del notebook **divisiones-de-entrenamiento-prueba-ibm-machine-l.ipynb**.

◆ Celda 1

```
python

!pip install pandas
!pip install numpy
!pip install matplotlib
!pip install scikit-learn
!pip install seaborn
```

🚩 ¿Qué hace esta celda?

Instala las bibliotecas necesarias usando **pip** (el sistema de gestión de paquetes de Python). El signo **!** es específico de los entornos tipo Jupyter Notebook y permite ejecutar comandos de consola desde una celda.

📦 Librerías instaladas:

- **pandas**: para manipulación y análisis de datos.
- **numpy**: para operaciones numéricas eficientes con arrays.
- **matplotlib**: para visualización de datos.
- **scikit-learn**: para implementar algoritmos de Machine Learning.
- **seaborn**: biblioteca de visualización basada en **matplotlib**, orientada a estadísticas.

Teoría: Estas bibliotecas forman el núcleo del stack de ciencia de datos en Python. En proyectos de ML, se utilizan para preparar, explorar, modelar y visualizar datos.

◆ Celda 2

```
python

# Surpress warnings:
def warn(*args, **kwargs):
    pass
import warnings
warnings.warn = warn
```

🔪 ¿Qué hace esta celda?

Esta celda **suprime los warnings** (advertencias) que Python pueda generar durante la ejecución del código, reemplazando la función `warnings.warn` por una función vacía que no hace nada.

🧠 Fundamento:

- `warnings` es un módulo que notifica al usuario sobre situaciones que no son errores, pero podrían requerir atención.
- Reemplazar `warnings.warn` con una función nula puede ser útil para limpiar la salida cuando los warnings son esperables o irrelevantes.

⚠️ **Advertencia:** Suprimir advertencias puede ocultar problemas reales en el código, por lo que se recomienda hacer esto solo en notebooks educativos o si se conoce bien el origen de los warnings.

◆ Celda 3

```
python

import pandas as pd
import numpy as np

# Import the data using the file path
data = pd.read_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-ML240EN-SkillsNetwork/labs/data/Ames_Housing_Sales.csv")
data.head()

print(data.shape)
```

🔪 ¿Qué hace esta celda?

1. Importa las bibliotecas `pandas` y `numpy`.
2. Carga un conjunto de datos en un DataFrame llamado `data` desde una URL (archivo CSV).
3. Muestra las primeras filas del dataset (con `data.head()`, aunque esta línea no tiene `print`).
4. Imprime la forma del dataset ((1379, 80)) → 1379 filas y 80 columnas.

🧠 Fundamentos:

- `pd.read_csv(...)`: Carga un archivo CSV (Comma Separated Values) como un DataFrame de `pandas`.
- `data.shape`: Devuelve una tupla (`n_filas`, `n_columnas`).

- `head()`: Muestra por defecto las primeras 5 filas del DataFrame, útil para conocer rápidamente la estructura del dataset.

Contexto del Dataset: El conjunto de datos de Ames Housing contiene información detallada de viviendas vendidas en Ames, Iowa. Es un dataset comúnmente utilizado para ejercicios de regresión.

✓ CONCLUSIÓN PARCIAL DEL FUNCIONAMIENTO

Las primeras tres celdas tienen como propósito:

1. Instalar las bibliotecas necesarias.
2. Silenciar advertencias para no contaminar la salida visual.
3. Cargar y explorar el conjunto de datos que se usará para aplicar modelos de aprendizaje automático (probablemente regresión o clasificación).

Establecen el entorno para el análisis posterior de Machine Learning con datos reales de viviendas.

CONCEPTOS TEÓRICOS ADICIONALES

1. DataFrame (pandas):

- Es una estructura bidimensional con etiquetas (columnas y filas), ideal para análisis tabular.
- Soporta múltiples tipos de datos y operaciones eficientes de filtrado, agrupamiento, agregación, etc.

2. Warnings vs Errores:

- Un *warning* no detiene el código pero advierte sobre un posible problema (como uso de funciones obsoletas).
- Un *error* detiene la ejecución.

3. CSV:

- Formato plano de texto para almacenar tablas donde los valores están separados por comas.
- Es uno de los formatos más comunes en ciencia de datos.

Puedo continuar con las siguientes celdas. ¿Quieres que siga?



continuar

Continuamos con el análisis detallado del notebook. Aquí van las explicaciones de las **celdas 4 a 6**.

◆ Celda 4

```
python

data.dtypes.value_counts()
```

📌 ¿Qué hace esta celda?

Cuenta cuántas columnas del DataFrame tienen cada tipo de dato.

🔍 Resultado:

```
go

object    43
float64    21
int64     16
```

🧠 Fundamento:

- **data.dtypes**: Devuelve una serie con los tipos de datos por columna.
- **value_counts()**: Cuenta la frecuencia de cada tipo.

Esto permite entender la composición del dataset: en este caso, 43 columnas categóricas (tipo **object**), 21 columnas numéricas continuas (**float64**) y 16 numéricas discretas (**int64**).

◆ Celda 5

```
python

# Select the object (string) columns
mask = data.dtypes == object
categorical_cols = data.columns[mask]
```

📌 ¿Qué hace esta celda?

Selecciona las columnas categóricas del dataset (tipo **object**) y guarda sus nombres en **categorical_cols**.

🧠 Fundamento:

- **mask**: Es una serie booleana que indica qué columnas son de tipo **object**.
- **data.columns[mask]**: Usa la máscara para filtrar los nombres de las columnas categóricas.

Esto es el paso previo a aplicar técnicas de codificación (encoding), como el **One-Hot Encoding**, que es común para trabajar con modelos de Machine Learning que requieren entrada numérica.

◆ Celda 6

python

```
# Determine how many extra columns would be created
num_ohc_cols = (data[categorical_cols]
                .apply(lambda x: x.nunique())
                .sort_values(ascending=False))

# No need to encode if there is only one value
small_num_ohc_cols = num_ohc_cols.loc[num_ohc_cols>1]

# Number of one-hot columns is one less than the number of categories
small_num_ohc_cols -= 1

# This is 215 columns, assuming the original ones are dropped.
# This is quite a few extra columns!
small_num_ohc_cols.sum()
```

🔪 ¿Qué hace esta celda?

1. Calcula el número de valores únicos por columna categórica (`nunique()`).
2. Filtra las columnas que tienen más de un valor distinto.
3. Calcula cuántas columnas **One-Hot Encoding** se crearían por cada una (sería $n - 1$).
4. Suma total: **204 columnas nuevas**.

🧠 Fundamentos clave:

- **One-Hot Encoding (OHE):**
 - Método para convertir variables categóricas a formato numérico.
 - Cada categoría se transforma en una columna binaria (0 o 1).
 - Para evitar colinealidad, se suelen crear $n-1$ columnas para una variable con n categorías.
- **¿Por qué $n-1$?**
 - Porque con $n-1$ columnas se puede inferir la categoría restante (evita la *trampa de la variable ficticia* o *dummy variable trap*).

Este análisis previo al OHE ayuda a anticipar la explosión dimensional. Pasar de 43 columnas categóricas a 204 columnas adicionales puede afectar rendimiento y sobreajuste.

✅ CONCLUSIÓN PARCIAL

Las celdas 4 a 6 se enfocan en **identificar y analizar las variables categóricas** del conjunto de datos:

- Determinan cuántas son.
- Calculan cuántas columnas adicionales se generarían al aplicar **One-Hot Encoding**.
- Evalúan si es necesario reducir la dimensionalidad o aplicar encoding selectivo.

Este paso es crucial en el **preprocesamiento** de datos antes de aplicar modelos de ML.

CONCEPTOS TEÓRICOS ADICIONALES

1. 🏷️ Variables Categóricas

- Representan categorías o etiquetas (como "madera", "metal", "cemento").
- No pueden usarse directamente en modelos que esperan valores numéricos.

2. 📊 One-Hot Encoding

- Técnica común de preprocesamiento.
- Produce variables binarias.
- Ejemplo: columna **Color** con categorías **Rojo**, **Verde**, **Azul** se convierte en:
 - **Color_Rojo**, **Color_Verde** → (**Azul** es inferido).

3. 🧠 Riesgo de Alta Dimensionalidad

- Agregar muchas columnas puede hacer que el modelo:
 - Tarde más en entrenar.
 - Tenga menor interpretabilidad.
 - Sea propenso al **sobreajuste**.

¿Deseas que continúe con las siguientes celdas?



continuar

A continuación, te explico en detalle las **celdas 7 a 9**, que se centran en **aplicar codificación a las variables categóricas** del dataset.

♦ Celda 7

```
python

from sklearn.preprocessing import OneHotEncoder, LabelEncoder

# Copy of the data
data_ohc = data.copy()

# The encoders
le = LabelEncoder()
ohc = OneHotEncoder()

for col in num_ohc_cols.index:

    # Integer encode the string categories
    dat = le.fit_transform(data_ohc[col]).astype(int)

    # Remove the original column from the dataframe
    data_ohc = data_ohc.drop(col, axis=1)

    # One hot encode the data--this returns a sparse array
    new_dat = ohc.fit_transform(dat.reshape(-1,1))
```

```
# Create unique column names
n_cols = new_dat.shape[1]
col_names = ['_'.join([col, str(x)]) for x in range(n_cols)]

# Create the new dataframe
new_df = pd.DataFrame(new_dat.toarray(),
                      index=data_ohc.index,
                      columns=col_names)

# Append the new data to the dataframe
data_ohc = pd.concat([data_ohc, new_df], axis=1)
```

🔗 ¿Qué hace esta celda?

Convierte todas las variables categóricas seleccionadas (`num_ohc_cols.index`) en variables binarias usando **Label Encoding + One-Hot Encoding**, y construye un nuevo DataFrame `data_ohc`.

Paso a paso:

1. Se hace una copia del dataset original.
2. Se instancian dos codificadores:
 - **LabelEncoder**: convierte categorías en enteros (requerido por **OneHotEncoder**).
 - **OneHotEncoder**: transforma enteros en vectores binarios (uno por categoría).
3. Para cada columna categórica:
 - Se codifica con enteros.
 - Se borra del DataFrame original.
 - Se aplica One-Hot Encoding.
 - Se renombran las columnas nuevas para que sean interpretables.
 - Se agregan al DataFrame `data_ohc`.

🧠 Fundamento técnico:

- **LabelEncoder** transforma por ejemplo `['red', 'blue', 'green']` → `[2, 0, 1]`.
- **OneHotEncoder** transforma eso en una matriz binaria:
`[2] → [0 0 1], [0] → [1 0 0]`, etc.

Esta combinación es necesaria porque **OneHotEncoder** de **scikit-learn** requiere valores numéricos como entrada.

◆ Celda 8

```
python

# Column difference is as calculated above
data_ohc.shape[1] - data.shape[1]
```

🔗 ¿Qué hace esta celda?

Calcula cuántas columnas nuevas se añadieron al DataFrame como resultado del One-Hot Encoding.

Resultado:

215

Esto confirma que se crearon 215 columnas nuevas a partir de las variables categóricas originales, como se había anticipado antes.

◆ Celda 9

```
python

print(data.shape[1])

# Remove the string columns from the dataframe
data = data.drop(num_ohc_cols.index, axis=1)

print(data.shape[1])
```

📌 ¿Qué hace esta celda?

1. Muestra cuántas columnas tiene el dataset original (80).
2. Elimina del DataFrame original las columnas categóricas que ya se codificaron.
3. Muestra el nuevo número de columnas (37), es decir, quedaron solo las columnas numéricas originales.

Esto deja al dataset **data** listo para usarse en modelos que no pueden manejar variables categóricas directamente.

✓ CONCLUSIÓN PARCIAL

Estas celdas realizaron el proceso completo de codificación de variables categóricas:

- Usaron **LabelEncoder** y **OneHotEncoder** de **scikit-learn**.
- Generaron un nuevo DataFrame **data_ohc** con 215 columnas adicionales binarias.
- Dejarán el conjunto **data** reducido a columnas numéricas, facilitando su uso en otros enfoques si se desea (como normalización, clustering, etc.).

Este paso es parte esencial del **preprocesamiento de datos**, donde transformamos datos brutos en una forma adecuada para ser procesada por algoritmos de Machine Learning.

CONCEPTOS TEÓRICOS ADICIONALES

1. LabelEncoder VS OneHotEncoder

- **LabelEncoder**: útil para árboles de decisión o cuando las categorías tienen un orden implícito.
- **OneHotEncoder**: necesario para modelos lineales o redes neuronales, ya que evita que el modelo interprete relaciones ordinales inexistentes entre categorías.

2. 🧠 Curse of Dimensionality

- Agregar muchas columnas puede dispersar los datos en el espacio y hacer más difícil el entrenamiento efectivo de algunos modelos.
- En esos casos, se recomienda usar técnicas de **reducción de dimensionalidad** como PCA.

¿Deseas que continúe con las siguientes celdas del notebook?



continuar

Vamos ahora con la **explicación detallada** de las **celdas 10 a 12**, donde se realiza la **división del conjunto de datos** en entrenamiento y prueba, tanto para la versión original como para la codificada.

◆ Celda 10

```
python

from sklearn.model_selection import train_test_split

y_col = 'SalePrice'

# Split the data that is not one-hot encoded
feature_cols = [x for x in data.columns if x != y_col]
X_data = data[feature_cols]
y_data = data[y_col]

X_train, X_test, y_train, y_test = train_test_split(X_data, y_data,
                                                    test_size=0.3, random_state=42)

# Split the data that is one-hot encoded
feature_cols = [x for x in data_ohc.columns if x != y_col]
X_data_ohc = data_ohc[feature_cols]
y_data_ohc = data_ohc[y_col]

X_train_ohc, X_test_ohc, y_train_ohc, y_test_ohc = train_test_split(X_data_ohc, y_data_ohc,
                                                                    test_size=0.3,
                                                                    random_state=42)
```

🚩 ¿Qué hace esta celda?

1. Separa la variable objetivo **SalePrice** de las variables predictoras.
2. Aplica **train-test split** para dividir el dataset en entrenamiento (70%) y prueba (30%).
3. Realiza esta operación dos veces:
 - Una con el dataset **original** (**data**) sin variables categóricas.
 - Otra con el dataset **codificado** (**data_ohc**) con One-Hot Encoding.

🧠 Fundamento técnico:

- **train_test_split**: divide aleatoriamente un dataset en subconjuntos de entrenamiento y prueba.
 - **test_size=0.3** → 30% para prueba, 70% para entrenamiento.
 - **random_state=42** → fija la semilla para garantizar reproducibilidad.
- La variable **SalePrice** es el **target** o **variable dependiente** que se quiere predecir.

Tener ambas versiones (**data** y **data_ohc**) permite comparar cómo influye la codificación en el rendimiento del modelo.

◆ Celda 11

```
python
```

```
# Compare the indices to ensure they are identical  
(X_train_ohc.index == X_train.index).all()
```

📌 ¿Qué hace esta celda?

Verifica si los índices de las filas de **X_train_ohc** y **X_train** son **idénticos**.

Resultado:

```
graphql
```

```
True
```

🧠 Importancia:

Asegurar que los conjuntos **X_train** y **X_train_ohc** tengan el mismo orden y los mismos registros permite:

- Hacer comparaciones directas entre modelos entrenados con ambas versiones.
- Garantizar que la separación entre entrenamiento y prueba es coherente en ambos casos.

◆ Celda 12

```
python
```

```
X_train
```

📌 ¿Qué hace esta celda?

Muestra el contenido del conjunto de entrenamiento **X_train** (sin codificación One-Hot). Este dataset incluye solo columnas numéricas originales.

Observaciones:

- 965 filas (70% de 1379).
- 36 columnas (las numéricas originales).
- Las columnas incluyen variables como: `1stFlrSF`, `BedroomAbvGr`, `TotalBsmtSF`, `YearBuilt`, etc.

✓ CONCLUSIÓN PARCIAL

Estas celdas dividen correctamente el conjunto de datos en entrenamiento y prueba, para ambas versiones:

- **Codificada** con One-Hot Encoding (`data_ohc`).
- **Numérica pura**, descartando categóricas (`data`).

Además, se verifica que ambas divisiones mantienen los mismos índices, garantizando consistencia para comparaciones posteriores.

CONCEPTOS TEÓRICOS ADICIONALES

1. Variable Objetivo (target)

En Machine Learning supervisado, es la variable que queremos predecir. En este caso: **SalePrice**.

2. Entrenamiento vs Prueba

- **Entrenamiento (70%)**: se usa para ajustar los parámetros del modelo.
- **Prueba (30%)**: se usa para evaluar el desempeño del modelo en datos nunca vistos.

3. Reproducibilidad

- `random_state`: garantiza que cada vez que ejecutes el código, la división sea la misma.
- Es fundamental para poder comparar resultados entre modelos de forma justa.

¿Quieres que continúe con las siguientes celdas del notebook?



continuar

Veamos en detalle las **celdas 13 a 15**, donde se comparan modelos de regresión lineal y se evalúa el efecto de diferentes técnicas de escalado y codificación.

◆ Celda 13

python

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

LR = LinearRegression()

# Storage for error values
error_df = list()

# Data that have not been one-hot encoded
LR = LR.fit(X_train, y_train)
y_train_pred = LR.predict(X_train)
y_test_pred = LR.predict(X_test)

error_df.append(pd.Series({'train': mean_squared_error(y_train, y_train_pred),
                          'test' : mean_squared_error(y_test, y_test_pred)},
                          name='no enc'))

# Data that have been one-hot encoded
LR = LR.fit(X_train_ohc, y_train_ohc)
y_train_ohc_pred = LR.predict(X_train_ohc)
y_test_ohc_pred = LR.predict(X_test_ohc)

error_df.append(pd.Series({'train': mean_squared_error(y_train_ohc, y_train_ohc_pred),
                          'test' : mean_squared_error(y_test_ohc, y_test_ohc_pred)},
                          name='one-hot enc'))

# Assemble the results
error_df = pd.concat(error_df, axis=1)
error_df
```

🔪 ¿Qué hace esta celda?

1. Entrena un modelo de **regresión lineal** con:
 - El dataset **sin codificación** (data)
 - El dataset **con One-Hot Encoding** (data_ohc)
2. Calcula el **Error Cuadrático Medio (MSE)** en entrenamiento y prueba.
3. Muestra un DataFrame comparativo.

📊 Resultado:

bash

	no enc	one-hot enc	
train	1.13e+09	3.17e+08	
test	1.37e+09	4.51e+19	← Δ

🔍 Observaciones:

- El modelo codificado (**one-hot enc**) tiene mejor rendimiento en entrenamiento.
- Pero el error en test es **exageradamente alto** ($\approx 10^{19}$), indicando **sobreajuste (overfitting)**.

🧠 Fundamento:

- El modelo con codificación **se ajusta demasiado a los datos de entrenamiento**, perdiendo capacidad de generalizar.

- Esto es típico cuando hay muchas variables y pocas observaciones.

◆ Celda 14

```
python

# Mute the setting with a copy warnings
pd.options.mode.chained_assignment = None
```

🔗 ¿Qué hace esta celda?

Desactiva advertencias de **pandas** relacionadas con **asignaciones encadenadas** (cuando se modifican subconjuntos de DataFrames).

Se hace para evitar que estas advertencias contaminen la salida del notebook.

◆ Celda 15

```
python

from sklearn.preprocessing import StandardScaler, MinMaxScaler, MaxAbsScaler

scalers = {'standard': StandardScaler(),
           'minmax': MinMaxScaler(),
           'maxabs': MaxAbsScaler()}

training_test_sets = {
    'not_encoded': (X_train, y_train, X_test, y_test),
    'one_hot_encoded': (X_train_ohc, y_train_ohc, X_test_ohc, y_test_ohc)}

# Get the list of float columns
mask = X_train.dtypes == float
float_columns = X_train.columns[mask]

# initialize model
LR = LinearRegression()

# iterate over all combinations and get the errors
errors = {}
for encoding_label, (_X_train, _y_train, _X_test, _y_test) in training_test_sets.items():
    for scaler_label, scaler in scalers.items():
        trainingset = _X_train.copy()
        testset = _X_test.copy()
        trainingset[float_columns] = scaler.fit_transform(trainingset[float_columns])
        testset[float_columns] = scaler.transform(testset[float_columns])
        LR.fit(trainingset, _y_train)
        predictions = LR.predict(testset)
        key = encoding_label + ' - ' + scaler_label + 'scaling'
        errors[key] = mean_squared_error(_y_test, predictions)

errors = pd.Series(errors)
print(errors.to_string())
```

🔗 ¿Qué hace esta celda?

Evalúa el impacto de **diferentes técnicas de escalado** sobre el rendimiento del modelo:

- Para los datasets:

- **not_encoded** (solo numérico)
- **one_hot_encoded** (con One-Hot Encoding)
- Aplica escaladores:
 - **StandardScaler**: normalización con media 0 y desviación estándar 1
 - **MinMaxScaler**: escala entre [0, 1]
 - **MaxAbsScaler**: escala entre [-1, 1] usando el valor absoluto máximo

Resultados clave:

```
nginx

not_encoded - standardscaling      ≈ 1.37e+09
not_encoded - minmaxscaling        ≈ 1.37e+09
not_encoded - maxabsscaling        ≈ 1.37e+09
one_hot_encoded - standardscaling  ≈ 1.68e+27 ← Δ descomunal
one_hot_encoded - minmaxscaling    ≈ 8.06e+09
one_hot_encoded - maxabsscaling    ≈ 8.06e+09
```

Conclusiones técnicas:

- El escalado **no mejora** significativamente el dataset sin codificar (numérico).
- En el dataset codificado:
 - **StandardScaler rompe el modelo** (error astronómico).
 - **MinMaxScaler** y **MaxAbsScaler** ayudan a controlar el sobreajuste.

Esto sugiere que **los datos codificados deben escalarse cuidadosamente**, y no todos los escaladores funcionan bien con variables binarias.

CONCLUSIÓN PARCIAL

Estas celdas completan la primera fase de modelado del notebook:

- Entrenan y evalúan un modelo lineal con y sin codificación.
- Diagnostican sobreajuste en datos codificados.
- Aplican diferentes técnicas de escalado para evaluar su efecto.

El mejor equilibrio entre rendimiento y generalización parece encontrarse usando **MinMaxScaler** o **MaxAbsScaler** con codificación One-Hot.

CONCEPTOS TEÓRICOS ADICIONALES

1. Regresión Lineal

- Modelo que ajusta una combinación lineal de variables predictoras para estimar un valor continuo.
- Fórmula: $\hat{y} = \beta_0 + \beta_1 x_1 + \dots + \beta_n x_n$

2. Error Cuadrático Medio (MSE)

- Métrica de error común para regresión:

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

- Penaliza fuertemente errores grandes (por el cuadrado).

3. Escalado

- Muchos algoritmos (como regresión lineal, redes neuronales, SVM) **funcionan mejor cuando los datos están escalados**.
- One-Hot Encoding produce variables binarias (0 o 1), que pueden desbalancear el modelo si no se normalizan bien con el resto de columnas.

¿Deseas que continúe con las últimas celdas? ¿O quieres una conclusión general del notebook hasta aquí?



continuar

Vamos con la **última celda del notebook**, ya que solo queda una más tras las analizadas previamente.

♦ Celda 16 (última)

```
python

# Looks like the minmaxscaler wins in this case.
```

¿Qué hace esta celda?

Es simplemente un **comentario** que resume los resultados obtenidos en las celdas anteriores:

"Parece que el **MinMaxScaler** gana en este caso."

CONCLUSIÓN GENERAL DEL NOTEBOOK

Este notebook realiza un análisis completo de cómo afecta la **codificación** y el **escalado** de características al desempeño de un modelo de **regresión lineal** aplicado al dataset de precios de viviendas de Ames.

Flujo general del código:

1. **Carga del dataset:** se cargaron datos reales de viviendas, con variables categóricas y numéricas.

2. Análisis de variables categóricas:

- Se identificaron 43 columnas categóricas.
- Se estimó la expansión dimensional por aplicar One-Hot Encoding (215 columnas nuevas).

3. Preprocesamiento:

- Se usaron **LabelEncoder** + **OneHotEncoder** para transformar las variables categóricas.
- Se generaron dos datasets: uno codificado (**data_ohc**) y otro numérico (**data**).

4. División de datos:

- Los datos fueron separados en entrenamiento (70%) y prueba (30%), de forma coherente entre ambas versiones.

5. Entrenamiento de modelos:

- Se entrenaron modelos de regresión lineal sobre ambos datasets.
- El modelo codificado sobreajustó los datos, mostrando un error exagerado en test.

6. Escalado de variables:

- Se compararon 3 técnicas: **StandardScaler**, **MinMaxScaler**, **MaxAbsScaler**.
- El escalado **MinMax** fue el que dio mejores resultados en el conjunto codificado, con menor error en test.

CONCEPTOS Y TEORÍA ADICIONAL

1. One-Hot Encoding y alta dimensionalidad

- Convierte variables categóricas en variables binarias.
- Puede generar cientos de columnas extra, lo que:
 - Aumenta el riesgo de sobreajuste.
 - Dificulta la interpretación del modelo.
 - Requiere escalado cuidadoso.

2. Escaladores:

Escalador	¿Qué hace?	Comentario
StandardScaler	Media = 0, Desvío = 1	Apto para modelos lineales, pero no ideal para binarios
MinMaxScaler	Escala a [0, 1]	✓ Mejor rendimiento en este caso
MaxAbsScaler	Escala a [-1, 1]	Bueno para datos dispersos o binarios

3. Error Cuadrático Medio (MSE)

- Cuantifica el promedio de los errores al cuadrado entre predicciones y valores reales.
- Muy sensible a errores grandes.

4. Regresión Lineal

- Modelo lineal simple, pero sensible al escalado, a la multicolinealidad y al ruido.

5. 🤖 Preprocesamiento como etapa crítica

- La preparación de datos (codificación, escalado) influye directamente en el rendimiento del modelo.
- Un mal preprocesamiento puede destruir la capacidad predictiva de un modelo, incluso si este es matemáticamente correcto.

✅ AVISO FINAL

NO HAY MÁS CELDAS DE CÓDIGO PARA EXPLICAR.

El notebook ha sido analizado completamente.

¿Te gustaría que prepare un resumen visual (gráfico) de los errores? ¿O deseas un texto que puedas usar como parte de un informe o presentación?



Podrías darme un resumen de toda la teoría de Machine Learning

¡Claro! Aquí tienes un **resumen completo y estructurado de la teoría de Machine Learning (ML)**. Lo divido en secciones para facilitar su comprensión:

📌 ¿Qué es Machine Learning?

Machine Learning es una subdisciplina de la inteligencia artificial (IA) que se enfoca en desarrollar algoritmos que pueden **aprender de datos** y **hacer predicciones o tomar decisiones** sin ser programados explícitamente para cada tarea.

🧠 Tipos de Aprendizaje

1. Aprendizaje Supervisado

- **Entradas + Salidas conocidas.**
- El modelo aprende a predecir la salida (target) a partir de las entradas.
- ♦ Ejemplos:
 - Clasificación (e.g. spam vs no spam)
 - Regresión (e.g. predecir precios)

2. Aprendizaje No Supervisado

- **Solo entradas, sin salidas.**

- El modelo descubre patrones ocultos en los datos.
- ♦ Ejemplos:
 - Clustering (agrupamiento)
 - Reducción de dimensionalidad (PCA, t-SNE)

3. Aprendizaje Semi-supervisado

- Mezcla de datos etiquetados y no etiquetados.
- Útil cuando etiquetar es costoso.

4. Aprendizaje por Refuerzo (Reinforcement Learning)

- El agente toma acciones en un entorno y aprende a maximizar una recompensa acumulada.
- ♦ Ejemplo: agentes que aprenden a jugar videojuegos.

Flujo General de un Proyecto de ML

1. Obtención y carga de datos
2. Limpieza y preprocesamiento
3. Análisis exploratorio (EDA)
4. Selección de características (feature selection/engineering)
5. División del dataset (train/test/validation)
6. Entrenamiento de modelos
7. Evaluación y métricas
8. Optimización (ajuste de hiperparámetros)
9. Despliegue y monitoreo

Modelos Comunes

Supervisado:

- Regresión lineal y logística
- Árboles de decisión y Random Forest
- Support Vector Machines (SVM)
- K-Nearest Neighbors (KNN)
- Redes neuronales

No supervisado:

- K-Means

- DBSCAN / HDBSCAN
- PCA / UMAP / t-SNE

Métricas de Evaluación

Clasificación:

- Accuracy, Precision, Recall, F1-Score
- Matriz de confusión
- AUC-ROC

Regresión:

- MAE (Error Absoluto Medio)
- MSE (Error Cuadrático Medio)
- RMSE (Raíz del MSE)
- R^2 (Coeficiente de determinación)

Preprocesamiento de Datos

- Limpieza de valores nulos o erróneos
- Escalado: `StandardScaler`, `MinMaxScaler`, `RobustScaler`
- Codificación: `OneHotEncoder`, `LabelEncoder`
- Reducción de dimensionalidad: PCA, UMAP

Selección y Optimización de Modelos

- **Cross-validation** (validación cruzada)
- **Grid Search / Random Search**
- **Técnicas de regularización:** Lasso (L1), Ridge (L2)

Overfitting vs Underfitting

- **Overfitting:** el modelo aprende demasiado bien el entrenamiento, pero falla en generalizar.
- **Underfitting:** el modelo es demasiado simple y no capta patrones reales.

Soluciones:

- Más datos
- Regularización

- Modelos más complejos (en underfitting) o más simples (en overfitting)



Aprendizaje Automático vs Estadística

- Ambos buscan inferencias sobre datos.
- ML prioriza **predicción** y rendimiento en test.
- Estadística clásica se enfoca más en **explicación** e inferencia.



Áreas Avanzadas

- **Deep Learning**: redes neuronales profundas (CNN, RNN, transformers)
- **AutoML**: automatización del pipeline de ML
- **Fairness y explicabilidad**: cómo entender y confiar en modelos complejos
- **MLOps**: integración y operación de modelos en producción

¿Quieres que prepare esto como un PDF, o que profundice en alguna sección específica (por ejemplo: algoritmos, métricas, deep learning)?