

ly6diplwf

December 4, 2025

1 CKD Dataset Preprocessing & Split (70/10/20)

```
[1]: # =====  
#   CKD Dataset Preprocessing & Split (70/10/20)  
# Author: Nasser Qanabta  
# Description:  
#   1) Load raw CKD CSV (Spanish headers + mixed categories)  
#   2) Translate column names and categorical values to English  
#   3) Clean data: duplicates, missing values, text normalization  
#   4) Encode categorical features (Label Encoding)  
#   5) Scale numeric features (StandardScaler)  
#   6) Create target column (Chronic_Kidney_Disease: 0/1)  
#   7) Split into Train (70%), Validation (10%), Test (20%)  
#   8) Save all outputs as CSV files  
# =====  
  
import pandas as pd  
import numpy as np  
import re  
from sklearn.model_selection import train_test_split  
from sklearn.preprocessing import LabelEncoder, StandardScaler  
from google.colab import files  
  
# -----  
# 1) Upload raw CKD dataset  
# -----  
print(" Please upload your FULL CKD dataset (CSV file with 15052 rows)...")  
uploaded = files.upload()  
filename = list(uploaded.keys())[0]  
  
# Read CSV  
df = pd.read_csv(filename)  
  
# -----  
# 2) Translate column names to English  
# -----  
translation_dict = {
```

```

    "Identificación del paciente": "Patient_ID",
    "Edad": "Age",
    "Presión arterial": "Blood_Pressure",
    "Gravedad específica de la orina": "Urine_Specific_Gravity",
    "Albúmina en orina": "Urine_Albumin",
    "Azúcar en orina": "Urine_Sugar",
    "Glóbulos rojos": "Red_Blood_Cells",
    "Células de pus": "Pus_Cells",
    "Grupos de células de pus": "Pus_Cell_Clumps",
    "Bacterias": "Bacteria",
    "Glucosa en sangre aleatoria": "Random_Blood_Glucose",
    "Urea en sangre": "Blood_Urea",
    "Creatinina sérica": "Serum_Creatinine",
    "Sodio en sangre": "Blood_Sodium",
    "Potasio en sangre": "Blood_Potassium",
    "Hemoglobina": "Hemoglobin",
    "Volumen corpuscular empaquetado": "Packed_Cell_Volume",
    "Recuento de glóbulos blancos": "WBC_Count",
    "Recuento de glóbulos rojos": "RBC_Count",
    "Apetito": "Appetite",
    "Insuficiencia renal crónica": "Chronic_Kidney_Disease"
}

df.rename(columns=translation_dict, inplace=True)

# -----
# 3) Translate categorical values to English
# -----
cat_translation = {
    "anormal": "abnormal",
    "presente": "present",
    "ausente": "absent",
    "bajo": "poor",
    "bueno": "good",
    "si": "yes",
    "no": "no"
}

# Apply translation to all object columns
for col in df.select_dtypes(include='object').columns:
    df[col] = df[col].replace(cat_translation)

# -----
# 4) Remove duplicates
# -----
df.drop_duplicates(inplace=True)

```

```

# -----
# 5) Handle missing values
#   - Numeric: fill with median
#   - Categorical: fill with mode
# -----
for col in df.columns:
    if df[col].dtype in ['float64', 'int64']:
        df[col] = df[col].fillna(df[col].median())
    else:
        df[col] = df[col].fillna(df[col].mode()[0])

# -----
# 6) Clean text in categorical columns
# -----
def clean_text(text):
    if not isinstance(text, str):
        return text
    # Lowercase
    text = text.lower()
    # Keep only allowed characters
    text = re.sub(r"[^a-zA-Z0-9\s.,!?'-]", '', text)
    # Remove extra spaces
    text = re.sub(r'\s+', ' ', text).strip()
    return text

for col in df.select_dtypes(include='object').columns:
    df[col] = df[col].apply(clean_text)

# -----
# 7) Create target column (0/1 for CKD)
#   Assuming 'yes' = CKD present, 'no' = not CKD
# -----
target_col = "Chronic_Kidney_Disease"

# Ensure target is text yes/no first
df[target_col] = df[target_col].astype(str).str.lower().str.strip()
df[target_col] = df[target_col].replace({'yes': 1, 'no': 0})

# -----
# 8) Separate features and target
# -----
X = df.drop(columns=[target_col])
y = df[target_col]

# -----
# 9) Encode categorical features (Label Encoding)
#   Note: For tree models, LabelEncoder is usually enough

```

```

# -----
categorical_cols = X.select_dtypes(include='object').columns

label_encoders = {}
for col in categorical_cols:
    le = LabelEncoder()
    X[col] = le.fit_transform(X[col])
    label_encoders[col] = le

# -----
# 10) Scale numeric features
#     (Tree models do not require scaling, but it helps if later
#     you use other models like Logistic Regression / SVM)
# -----
numeric_cols = X.select_dtypes(include=['int64', 'float64']).columns

scaler = StandardScaler()
X[numeric_cols] = scaler.fit_transform(X[numeric_cols])

# -----
# 11) Save full processed dataset (features + target)
# -----
df_final = X.copy()
df_final[target_col] = y

df_final.to_csv("CKD_cleaned_dataset_EN.csv", index=False)

print(" Preprocessing complete.")
print(" File saved:")
print("   - CKD_cleaned_dataset_EN.csv")

# -----
# 12) (Optional) Download file to your local machine
# -----
from google.colab import files
files.download("CKD_cleaned_dataset_EN.csv")

```

Please upload your FULL CKD dataset (CSV file with 15052 rows)...

<IPython.core.display.HTML object>

Saving Dataset_insuficiencia_renal_crnica.csv to
Dataset_insuficiencia_renal_crnica.csv

/tmp/ipython-input-3250016622.py:119: FutureWarning: Downcasting behavior in
`replace` is deprecated and will be removed in a future version. To retain the
old behavior, explicitly call `result.infer\_objects(copy=False)`. To opt-in to
the future behavior, set `pd.set\_option('future.no\_silent\_downcasting', True)`
df[target_col] = df[target_col].replace({'yes': 1, 'no': 0})

```
Preprocessing complete.  
File saved:  
  - CKD_cleaned_dataset_EN.csv  
<IPython.core.display.Javascript object>  
<IPython.core.display.Javascript object>
```