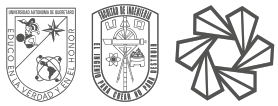


UNIVERSIDAD AUTÓNOMA DE QUERÉTARO
FACULTAD DE INGENIERÍA
DIVISIÓN DE INVESTIGACIÓN Y POSGRADO

# Machine Learning: Práctica 2. Imputación de datos.

*Alumno:*
Ing. Enrique Mena Camilo

*Profesor:*
Dr. Marco Antonio Aceves Fernández

Marzo 2023

# Índice

# 1. Objetivo

Desarrollar un conjunto de herramientas propias que sirvan para realizar imputación de datos a atributos categóricos y continuos.

Se deberá implementar como mínimo dos métodos de imputación, así como procurar usar por lo menos 1 atributo categórico y 1 atributo continuo.

# 2. Introducción

Es de conocimiento general que todo modelo de inteligencia artificial es tan bueno como los datos con los que fue entrenado.

Dentro de los problemas más comunes que pueden presentarse en los datos se destacan: datos faltantes, problemas de cardinalidad irregular y valores atípicos [1].

Hablando específicamente del problema de datos faltantes, podemos encontrarnos con diversas causas de dicho problema, desde errores en el sistema de adquisición de datos hasta errores humanos [1].

Para solucionar el problema de datos faltantes existen diversos métodos, desde el proceso de eliminar las instancias que cuenten con datos faltantes o bien eliminar los atributos, sin embargo, cuando la cantidad de datos faltantes representa menos del 30 % del total de instancias del atributo, suele ser prudente utilizar algún método de imputación de datos [1].

# 3. Marco teórico

## 3.1. Técnicas de imputación por información externa o deductiva

Consisten en deducir los datos faltantes mediante reglas pre-establecidas que contemplan a instancias completas [1].

## 3.2. Técnicas deterministas de imputación

Útiles cuando de acuerdo a las mismas condiciones de los datos se producen las mismas respuestas.

- **Imputación por regresión**. Se ajusta un modelo lineal o polinomial usando atributos sin datos faltantes. Los datos faltantes se toman del ajuste resultante de la regresión [1].
- **Imputación de la media (o moda)**. Los datos faltantes se llenan con la media (o moda en el caso de atributos cualitativos) de las instancias no faltantes [1].
- **Imputación por media de clases**. Se calcula la media (o moda) de las instancias que tienen valor por cada clase y se llena el valor faltante para cada una de las clases [1].
- **Imputación por vecino más cercano**. Se calcula la distancia entre la instancia a imputar y los datos que tienen valor establecido. El dato más cercano será el utilizado para imputar la instancia faltante.

## 3.3. Técnicas estocásticas de imputación

Se definen como aquellas técnicas de imputación que al repetirse bajo las mismas condiciones producen resultados diferentes.

- **Imputación aleatoria**. Se toman las posibles observaciones del atributo con datos faltantes, se selecciona un valor dentro del rango existente y se llena el dato faltante con dicha elección aleatoria [1].
- **Imputación secuencial**. Consiste en tomar de manera secuencial los datos existentes para reemplazar los datos faltantes. Se toma de manera aleatoria un dato existente y se utiliza dicho valor para reemplazar el primer dato faltante, posteriormente se toma el siguiente valor existente y se utiliza para reemplazar el siguiente dato faltante, el proceso se repite hasta haber llenado todos los datos faltantes [1].

# 4. Materiales y métodos

Para el desarrollo de esta práctica se utilizó el lenguaje de programación Python en su versión 3.10, con el que se diseñó un script para cumplir los objetivos de la práctica.

## 4.1. Conjunto de datos utilizado

El conjunto de datos utilizado para esta práctica consta de una colección cuyo objetivo es la predicción de un derrame cerebral (stroke, en inglés). Dicho conjunto de datos fue obtenido de la plataforma Kaggle, y consta de 10 atributos de diversos tipos y 1 variable objetivo, teniendo un total de *40,910* instancias.

## 4.2. Generación de datos faltantes

Originalmente, el conjunto de datos utilizado solamente contaba con 3 instancias faltantes de un único atributo, por lo que se diseñó un script en Python que borra de forma aleatoria un porcentaje dado de datos del conjunto de datos.

## 4.3. Procedimiento para imputación de datos

Teniendo el conjunto de datos listo para aplicar los métodos de imputación, se procedió a realizar la siguiente lista de pasos:

1. Estimar el porcentaje de datos faltantes.
2. Generar histogramas de todas los atributos del conjunto de datos.
3. Generar mapa de calor de correlación del conjunto de datos.
4. Elegir técnica de imputación adecuada para cada atributo.
5. Realizar imputación de datos faltantes.
6. Generar histogramas de todos los atributos del conjunto de datos con los datos imputados.

Dichos pasos fueron implementados dentro de un script de Python que implementa los métodos de imputación elegidos, así como generar gráficos y mostrar resultados en terminal.

# 5. Resultados

## 5.1. Porcentaje de datos faltantes

Se determinó que el conjunto de datos utilizado cuenta con 5 atributos con datos faltantes, todos con un porcentaje menor al 30 %, por lo que se puede aplicar técnicas de imputación a todos los atributos con datos faltantes. La Tabla 1 muestra el detalle de los datos faltantes encontrados.

Tabla 1: Porcentaje de datos faltantes por atributo.

| Columna | Total de instancias | Instancias con datos faltantes | Porcentaje de datos faltantes |
|---|---|---|---|
| sex | 40910 | 3 | 0.01 % |
| age | 40910 | 0 | 0.00 % |
| hypertension | 40910 | 5676 | 16.11 % |
| heart_disease | 40910 | 0 | 0.00 % |
| ever_married | 40910 | 0 | 0.00 % |
| work_type | 40910 | 5675 | 16.11 % |
| Residence_type | 40910 | 5704 | 16.20 % |
| avg_glucose_level | 40910 | 0 | 0.00 % |
| bmi | 40910 | 5720 | 16.25 % |
| smoking_status | 40910 | 0 | 0.00 % |
| stroke | 40910 | 0 | 0.00 % |

## 5.2. Histogramas con datos originales

Observando los histogramas de todas las columnas del conjunto de datos (mostrados en la Figura 1 podemos observar lo siguiente de los atributos con datos faltantes.

- **sex, hypertension, Residence_type**
  - Los datos faltantes son categóricos.
  - Los datos faltantes sólo pueden tomar los valores 0 o 1.
- **work_type**
  - Los datos faltantes son categóricos.
  - Los datos faltantes sólo pueden tomar los valores 1, 2, 3, o 4.

- **bmi**
  - Los datos faltantes son continuos.

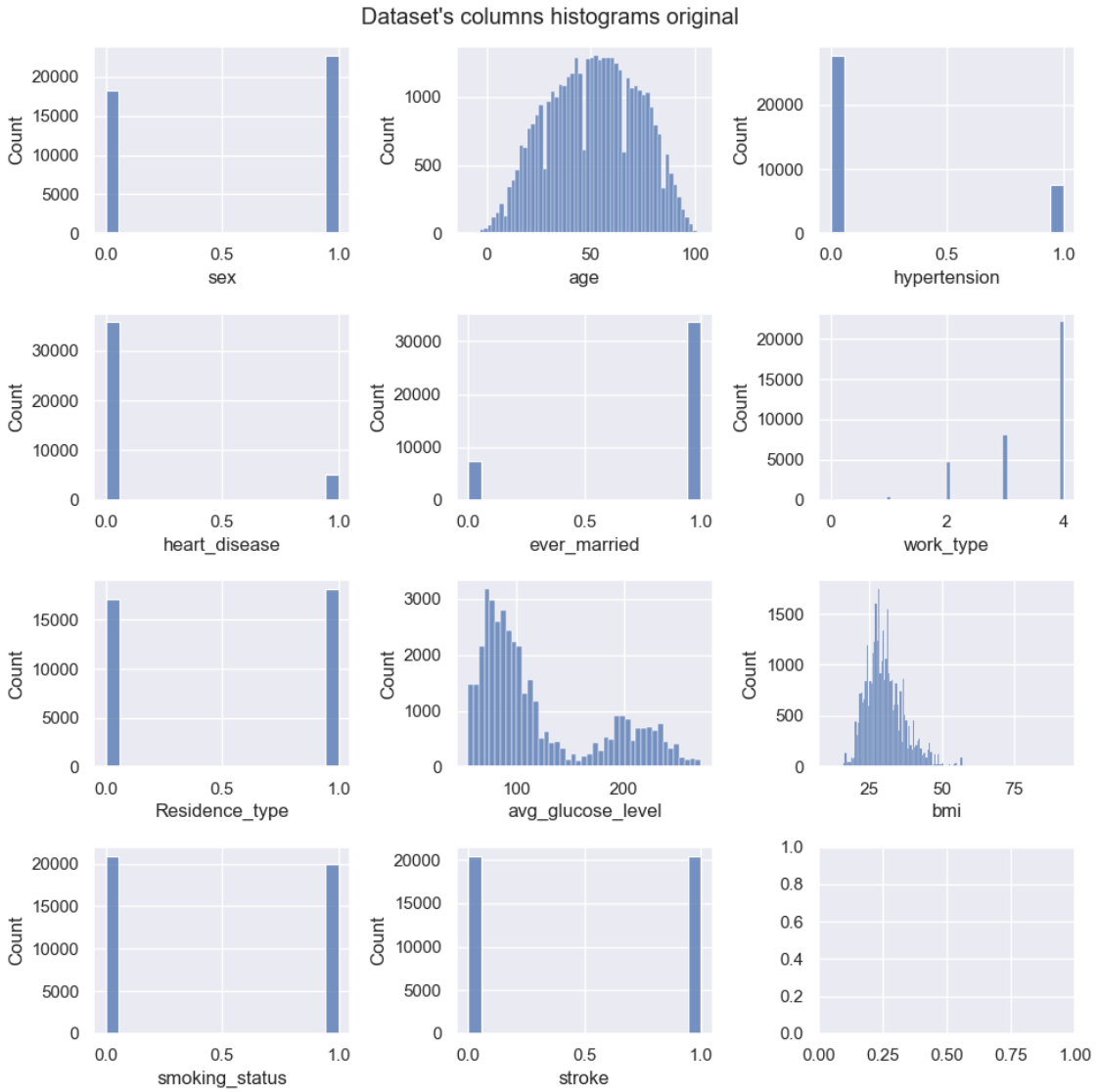



Figura 1: Histogramas de datos originales.

## 5.3. Mapa de calor de correlación

Al analizar el mapa de calor de correlación (Figura 2) se puede observar lo siguiente:

- Los atributos *work_type* y *Residence_type* presentan índices de correlación cercanos a cero, por lo que la *imputación aleatoria* podría aplicarse correctamente.
- El atributo *hypertension* consigue su mayor índice de correlación con la variable objetivo (*stroke*), por lo que una imputación por *moda de clases* es una elección apropiada.
- El atributo *bmi* consigue su mayor índice de correlación con el atributo *avg_glucose_level*, y dado que ambos son atributos de tipo continuo la *imputación por correlación* presenta una alternativa tentadora.

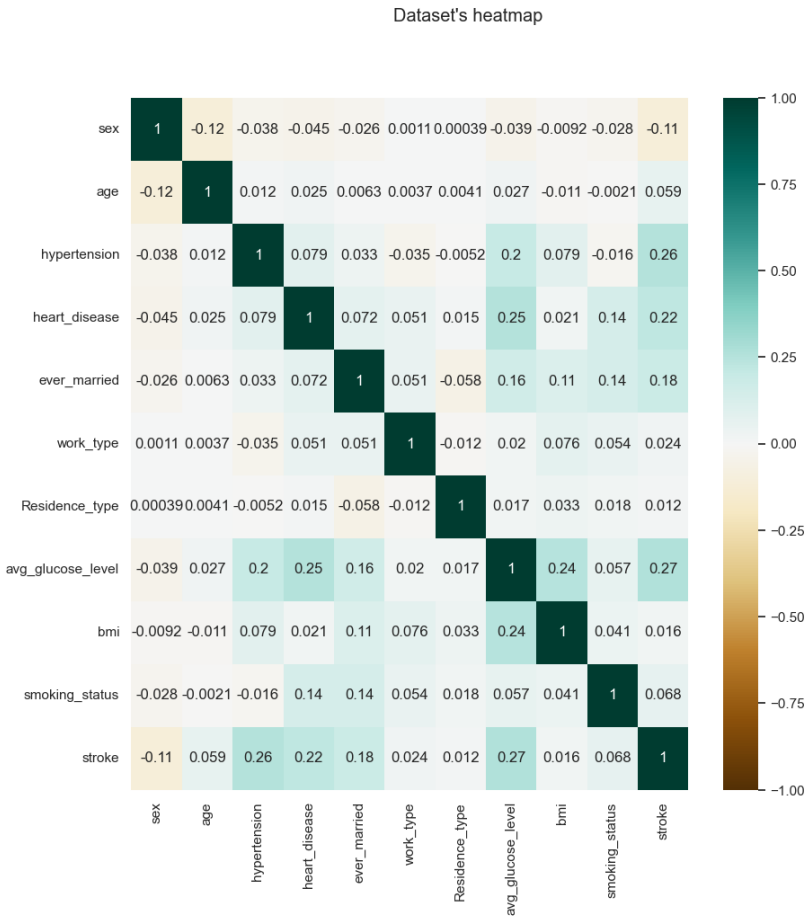



Figura 2: Mapa de calor de correlación entre columnas del conjunto de datos.

## 5.4. Histogramas con datos imputados

Prestando especial atención a los atributos que se sometieron a métodos de imputación, se puede observar que el procedimiento no ha cambiado la forma de la distribución de los datos, lo cuál es un indicador de una correcta implementación de los métodos de imputación.

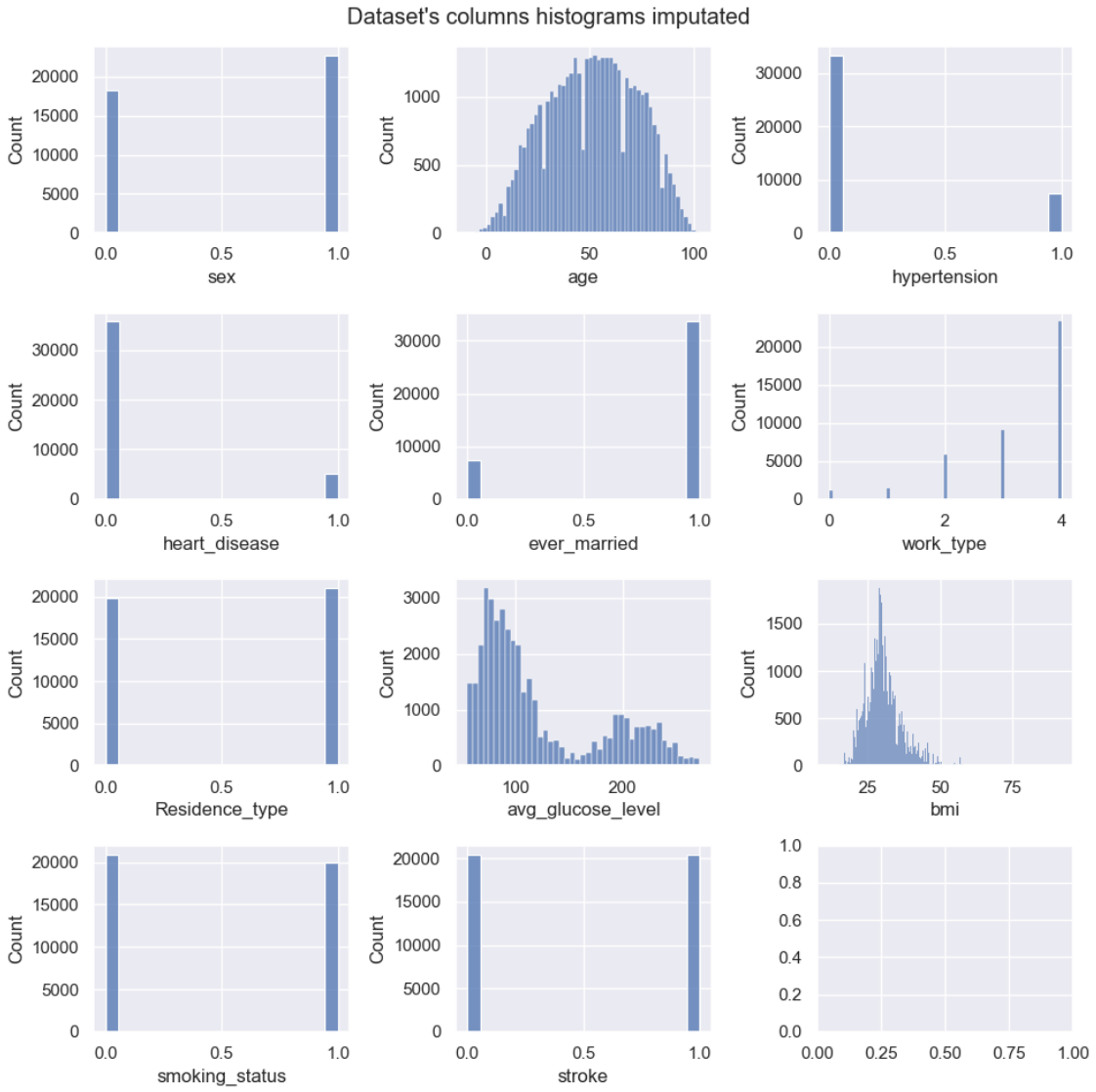


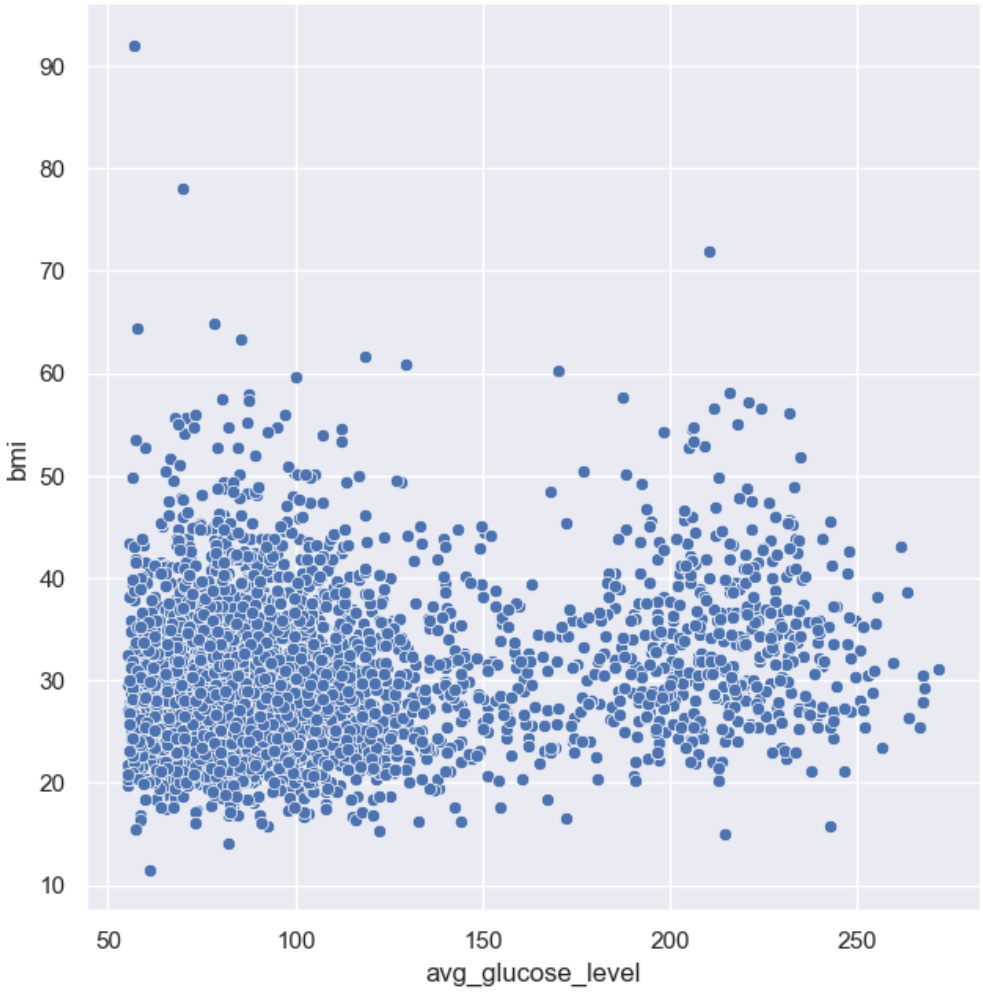

Figura 3: Histogramas de datos originales con datos imputados.

# 6. Conclusiones

El manejo de datos faltantes siempre presenta un gran reto dentro del área de inteligencia artificial. Los métodos analizados e implementados en esta práctica han demostrado ser útiles para llenar estos espacios vacíos y no cambiar la distribución original de los datos.

Hablando específicamente de cada uno de los atributos trabajados tenemos:

- **sex**. Dado a que solamente había 3 datos faltantes, fue bastante claro el implementar un método como la imputación por moda. Habría que investigar si dicho método es recomendable solamente ante ciertos rangos de datos faltantes.
- **hypertension**. Considerando que el índice de correlación máximo obtenido coincide con la variable objetivo, fue fácil imaginar que la imputación por media/moda de clases sería ideal, sin embargo al prestar atención al método podemos observar que la distribución de los posibles valores de este atributo es aproximadamente de 3:1, por lo que el método de moda de clase rellenó todos los faltantes con la clase 0, mismo resultado se hubiera obtenido con una imputación por moda y hubiera disminuido la complejidad computacional.
- **workt_type y Residence_type**. El método de imputación aleatoria logró mantener de forma adecuada la distribución de estas variables, comportamiento que se atribuye a la claras diferencias en la frecuencia de cada una de las observaciones de cada atributo, es muy probable que utilizar este método en atributos con mayor resolución no será eficaz.
- **bmi**. Considerar que el mayor índice de correlación fue con el atributo *avg_glucose_level* fue de ayuda para lograr estimar una línea de tendencia entre ambos atributos, sin embargo, dicho valor de correlación apenas fue de 0,24, lo cuál es un claro indicador de que no existe una relación lineal entre ambos atributos (afirmación que se confirma al generar un gráfico de dispersión entre ambos atributos (Figura **??**)). Existe la posibilidad de que utilizar algún tipo de regresión no lineal pueda mejorar el rendimiento de este método. A pesar de sus limitantes, es destacable el hecho de que se logró conservar la forma de la distribución de los datos originales.


bmi
90
80
70
60
50
40
30
20
10
50
100
150
200
250
avg_glucose_level

# Referencias bibliográficas

[1] M. A. Aceves Fernández, *Inteligencia artificial para programadores con prisa*. Universo de Letras, 2021, ISBN: 9788418854613.

# A. Código documentado

El código completo y funcional se puede encontrar anexo en el archivo *zip* compartido en conjunto con este reporte, así como en las secciones anexas.

## A.1. DeleteData.py

Script implementado para generar datos faltantes.

```python
#!/usr/bin/env python3
"""Script to delete data from data set.
"""

import numpy as np
import pandas as pd

from typing import Union

__author__ = "Enrique Mena Camilo"
__email__ = "enriquemece97@gmail.com"


def delete_data(dataset: pd.DataFrame, percentage: float = 0.15, to_delete: Union[None, list] = None):
    """Performs random data deletion.

    :param dataset: Data set to perform data deletion.
    :param percentage: Percentage of deletion
    :param to_delete: Columns to delete.
    :return pd.DataFrame: Data set with deleted data.
    """
    if percentage <= 0 or percentage >= 1:
        raise ValueError("'percentage' must be more than 0 and less than 1")

    dataset = dataset.copy()
    total_samples = dataset.shape[0]
    to_delete_samples = int(total_samples*percentage)

    if to_delete is None:
        to_delete = dataset.columns.to_list()

    for feature in to_delete:
        delete_samples = np.random.randint(0, total_samples, size=to_delete_samples)
        dataset.loc[delete_samples, feature] = np.nan

```

```
    return dataset


if __name__ == "__main__":
    data = pd.read_csv("../data/stroke.csv")
    features_to_delete = data.drop(columns=["sex", "age", "stroke", "
    heart_disease", "avg_glucose_level",
                                            "ever_married", "
    smoking_status"]).columns.to_list()
    data_deleted = delete_data(data, to_delete=features_to_delete)
    data_deleted.to_csv("../data/stroke_nan.csv", index=False)
```

## A.2. Practica2_EnriqueMenaCamilo.py

Script implementado para el desarrollo de la práctica.

```
#!/usr/bin/env python3
"""Machine learning: Practice 2.
Implement data imputation methods for categorical and continuous
    attributes.
By: Enrique Mena Camilo.
"""

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from tqdm import tqdm
from sklearn.preprocessing import MinMaxScaler

sns.set_theme(style="darkgrid")

__author__ = "Enrique Mena Camilo"
__email__ = "enriquemece97@gmail.com"


def predict_linear_regression(x_data: np.ndarray, w_data: np.ndarray) ->
    np.ndarray:
    """Gets predictions using the linear regression method.

    :param np.ndarray x_data: Data with which the prediction will be made.
    :param np.ndarray w_data: Weights obtained from linear regression
    training.
    :return np.ndarray: Results obtained from the prediction.
    """
    x_data = np.append(np.ones((len(x_data), 1)), x_data, axis=1)
    return np.dot(x_data, w_data)
```

```python


def train_linear_regression(x_data: np.ndarray, y_data: np.ndarray, lr: float = 0.1, it: int = 1000) -> np.ndarray:
    """Fit linear regression model.

    :param np.ndarray x_data: Training data.
    :param np.ndarray y_data: Target values.
    :param float lr: Learning rate. Defaults to 0.1.
    :param int it: Iterations. Defaults to 1000.
    :return np.ndarray: Weights from linear regression.
    """
    def cost(x_: np.ndarray, y_: np.ndarray, w_: np.ndarray) -> float:
        """MSE as cost function.

        :param np.ndarray x_: Attributes values.
        :param np.ndarray y_: Target values
        :param np.ndarray w_: Weights values.
        :return float:
        """
        return 1/(2 * len(y_)) * np.dot((np.dot(x_, w_) - y_).T, (np.dot(x_, w_) - y_))

    x_data, y_data = x_data.copy(), y_data.copy()

    x_data = np.append(np.ones((len(x_data), 1)), x_data, axis=1)
    x_len = len(x_data)
    theta = np.ones((x_data.shape[1], 1))

    for _ in tqdm(range(it)):
        theta = theta - lr * (1/x_len) * np.dot(x_data.T, np.dot(x_data, theta) - y_data)
    print(f"Final cost: {cost(x_data, y_data, theta)}")

    return theta


def target_mean_imputation(dataset: pd.DataFrame, feature: str, target: str, feature_type: str = "quanti") -> pd.DataFrame:
    """Performs data imputation using the target mean method.

    :param pd.DataFrame dataset: Full dataset.
    :param str feature: Feature name to imputate.
    :param str target: Target name.
    :param str feature_type: One of ['quanti', 'quali'].
    :return pd.DataFrame: Feature with imputated data.
    """
    to_fill = dataset[feature].copy()
```

```
74
75      for class_ in dataset[target].unique():
76          if feature_type == "quali":
77              estimated = to_fill.loc[dataset[target] == class_].mode()[0]
78          elif feature_type == "quanti":
79              estimated = to_fill.loc[dataset[target] == class_].mean()
80          else:
81              raise ValueError("'feature_type' must be one of ['quanti', '
    quali']")
82
83          to_fill.loc[(dataset[target] == class_) & (to_fill.isna())] =
    estimated
84
85      return to_fill
86
87
88  def random_imputation(dataset: pd.DataFrame, feature: str, feature_type:
    str = "quanti") -> pd.DataFrame:
89      """Performs data imputation using the random method.
90
91      :param pd.Dataframe dataset: Full dataset.
92      :param str feature: Feature name to imputate.
93      :param str feature_type: One of ['quanti', 'quali'].
94      :return pd.DataFrame: Feature with imputated data.
95      """
96      to_fill = dataset[feature].copy()
97      nan_count = to_fill.isna().sum()
98
99      if feature_type == "quanti":
100         estimated = np.random.uniform(low=to_fill.min(), high=to_fill.max
    ()+1, size=nan_count)
101     elif feature_type == "quali":
102         estimated = np.random.randint(low=to_fill.min(), high=to_fill.max
    ()+1, size=nan_count)
103     else:
104         raise ValueError("'feature_type' must be one of ['quanti', 'quali
    ']")
105
106     estimated = pd.Series(estimated, index=to_fill.loc[to_fill.isna()].
    index.to_list())
107     return to_fill.fillna(estimated)
108
109
110 class DatasetNanHandler:
111     def __init__(self, dataset: pd.DataFrame, target: str):
112         self.dataset = dataset.copy()
113         self.columns = dataset.columns.to_list()
114         self.target = target
```

```
115         self.features = dataset.drop(columns=[target]).columns.to_list()
116 
117         self.nan_count = None
118 
119     def get_nan_count(self, rebuild: bool = False) -> pd.DataFrame:
120         """Build information about missing values in the data set.
121 
122         :param bool rebuild: Flag to indicate whether to use previous data or recalculate. Defaults to False.
123         :return pd.DataFrame: DataFrame with information about nan values around full dataset.
124         """
125         if self.nan_count is None or rebuild:
126             nan_count = list()
127             for column in self.columns:
128                 nan_count.append({
129                     "column": column,
130                     "total_count": self.dataset[column].shape[0],
131                     "non_nan_count": self.dataset[column].count(),
132                     "nan_count": self.dataset[column].isna().sum(),
133                     "nan_percentage": round(self.dataset[column].isna().sum()/self.dataset[column].count(), 4)
134                 })
135 
136             self.nan_count = pd.DataFrame(nan_count).set_index("column")
137         return self.nan_count
138 
139     def get_histograms(self, shape: tuple, sufix: str = None):
140         """Build column's histograms in subplots matrix.
141 
142         :param tuple shape: Subplots shape.
143         :param str sufix: Sufix in the png file. Defaults to None.
144         """
145         fig, axes = plt.subplots(shape[0], shape[1])
146         fig.suptitle(f"Dataset's columns histograms {sufix}")
147         fig.set_size_inches(10, 10)
148         sufix = f"_{sufix}" if sufix else ""
149 
150         for ax, column in zip(axes.flatten(), self.columns):
151             sns.histplot(data=self.dataset, x=column, ax=ax)
152             ax.grid(True)
153         plt.tight_layout()
154         plt.savefig(f"./figures/dataset_histogram{sufix}.png", bbox_inches='tight')
155         plt.show()
156 
157     def get_heatmap(self):
158         """Build dataset correlation heatmap.
```

```
159         """
160         fig, ax = plt.subplots(1)
161         fig.suptitle(f"Dataset's heatmap")
162         fig.set_size_inches(10, 10)
163         sns.heatmap(data=self.dataset.corr(), annot=True, cmap='BrBG',
    vmin=-1, vmax=1)
164         plt.savefig(f"./figures/dataset_heatmap.png", bbox_inches='tight')
165         plt.show()
166 
167     def fill_nan(self, method: str, feature: str, **kwargs):
168         """Perform data imputation according to the choiced method.
169 
170         :param str method: Once of ['mean', 'mode', 'target_mean', 'random
    ', 'regression']
171         :param str feature: Feature name to imputate.
172         :param kwargs:
173         :return:
174         """
175         if method == "mean":
176             estimated = self.dataset[feature].mean()
177             self.dataset[feature].fillna(estimated, inplace=True)
178             return
179 
180         if method == "mode":
181             estimated = self.dataset[feature].mode()[0]
182             self.dataset[feature].fillna(estimated, inplace=True)
183             return
184 
185         if method == "target_mean":
186             estimated = target_mean_imputation(self.dataset, feature, self
    .target, kwargs.get("feature_type", "quali"))
187             self.dataset[feature] = estimated
188             return
189 
190         if method == "random":
191             estimated = random_imputation(self.dataset, feature, kwargs.
    get("feature_type", "quali"))
192             self.dataset[feature] = estimated
193             return
194 
195         if method == "regression":
196             if "predictors" in kwargs:
197                 predictors = kwargs["predictors"]
198             else:
199                 raise TypeError("Missing 1 required keyword-only argument:
     'predictors'")
200 
201             # Prepare predictors and target
```

```
202             x = self.dataset[self.dataset[feature].notnull()][predictors].
    to_numpy()
203             y = self.dataset[self.dataset[feature].notnull()][feature].
    to_numpy()
204             y = y.reshape(y.shape + (1,))
205 
206             # Normalize predictors
207             scaler = MinMaxScaler()
208             scaler.fit(x)
209             x = scaler.transform(x)
210 
211             # Train regression
212             weights = train_linear_regression(x, y)
213 
214             # Fill NaN values
215             x_nan = self.dataset[self.dataset[feature].isnull()][
    predictors]
216             to_fill_index = x_nan.index.to_list()
217             x_nan = scaler.transform(x_nan.to_numpy())
218             estimated = predict_linear_regression(x_nan, weights)[:, 0]
219             self.dataset.loc[to_fill_index, feature] = estimated
220             return
221 
222         raise ValueError("Invalid 'method'")
223 
224 
225 if __name__ == "__main__":
226     data = pd.read_csv("../data/stroke_nan.csv")
227     data_handler = DatasetNanHandler(data, target="stroke")
228 
229     print("====NaN values count (original)====")
230     print(data_handler.get_nan_count())
231 
232     print("====Histograms (original)====")
233     data_handler.get_histograms((4, 3), sufix="original")
234 
235     print("====Heatmap====")
236     data_handler.get_heatmap()
237 
238     print("====Mode imputation===")
239     data_handler.fill_nan("mode", "sex")
240 
241     print("====Target mean imputation====")
242     data_handler.fill_nan("target_mean", "hypertension", feature_type="
    quali")
243 
244     print("====Random imputation====")
```

```
245     data_handler.fill_nan("random", "Residence_type", feature_type="quali"
        )
246     data_handler.fill_nan("random", "work_type", feature_type="quali")
247
248     print("====Regression imputation====")
249     data_handler.fill_nan("regression", "bmi", predictors=["
        avg_glucose_level"])
250
251     # bmi vs glucose dispersion
252     ax = sns.pairplot(data, y_vars=["bmi"], x_vars=["avg_glucose_level"])
253     ax.fig.set_size_inches(8, 8)
254     plt.savefig("./figures/bmi_glucose.png", bbox_inches='tight')
255     plt.show()
256
257     print("====NaN values count (after imputation)====")
258     print(data_handler.get_nan_count(rebuild=True))
259
260     print("====Histograms (imputated)====")
261     data_handler.get_histograms((4, 3), sufix="imputated")
```