Tutorial: Estimación de precios de viviendas

1. **Configuración del Ambiente**

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

pandas: nos permite manejar tablas de datos.

numpy: facilita cálculos con números y arreglos.

matplotlib.pyplot: se usa para hacer gráficos.

seaborn: mejora y simplifica la creación de gráficos estadísticos.

Con esto, estamos listos para trabajar con datos y visualizarlos de manera efectiva.

2. **Exploración de Datos**

pd.read_csv(): Esta función carga un archivo CSV (que es como una tabla de datos) y lo convierte en un DataFrame de pandas. Así, podemos trabajar fácilmente con los datos en formato tabular.

```
data = pd.read_csv('California_Houses.csv')
data.info()
```

data.info(): Nos da un resumen del DataFrame. Muestra:

- El número de filas y columnas.
- Los tipos de datos (números, texto, etc.).
- Cuántos valores nulos o faltantes hay en cada columna.

```
data.describe()
```

data.describe(): Nos da estadísticas básicas para las columnas numéricas del DataFrame. Esto incluye:

- **count**: Número total de valores no nulos.
- **mean**: Promedio de los valores.
- **std**: Desviación estándar (qué tanto varían los datos).
- **min**: Valor mínimo.
- **25%, 50%, 75%**: Los percentiles (25%, mediana o 50%, y 75%).
- **max**: Valor máximo.

Ahora vamos a generar un **mapa de calor** basado en la **matriz de correlación** de los datos. Aquí te explico paso a paso las instrucciones y la utilidad de esta visualización:

2.1. Correlación de variables

¿Qué es una matriz de correlación?

- Una matriz de correlación muestra la relación entre las diferentes variables de un conjunto de datos.
- Cada valor en la matriz indica qué tan fuerte es la correlación entre dos variables. La correlación puede variar entre **-1 y 1**:
    - **1**: Correlación perfecta positiva (si una variable sube, la otra también).
    - **0**: No hay correlación (las variables no tienen relación).
    - **-1**: Correlación perfecta negativa (si una variable sube, la otra baja).

¿Por qué es útil?

- Identifica **relaciones fuertes** o patrones entre variables, lo que puede ayudarte a entender qué variables están estrechamente conectadas.
- Es una herramienta clave para evitar la multicolinealidad (cuando dos o más variables están muy relacionadas), que puede afectar los resultados de los modelos predictivos.

¿Cómo leer el mapa de calor?

- Los colores representan el grado de correlación:
    - **Rojo** indica correlación negativa.

    - **Azul** indica correlación positiva.
    - **Celdas cercanas a 0** serán de color neutro.
- Los números dentro de cada celda indican el valor exacto de la correlación entre dos variables específicas.

¿Cómo generar un mapa de calor de matriz de correlación?

- **Calcular la matriz de correlación**:

  Usa el método .corr() del DataFrame para obtener la matriz de correlación entre las variables numéricas del conjunto de datos.

- **Configurar el tamaño del gráfico**:

  Utiliza plt.figure(figsize=(ancho, alto)) de la librería matplotlib para ajustar el tamaño del gráfico.

- **Crear el mapa de calor**:

  Emplea sns.heatmap() del paquete seaborn para generar el mapa de calor a partir de la matriz de correlación. Configura una paleta de colores con el parámetro cmap y ajusta el grosor de las líneas entre celdas con linewidths.

- **Mostrar los valores de correlación**:

  Usa el parámetro annot=True dentro de sns.heatmap() para que los valores numéricos de la correlación aparezcan en cada celda del gráfico.

- **Añadir un título**:

  Con el método plt.title() de matplotlib, puedes añadir un título descriptivo al gráfico.

- **Mostrar el gráfico**:

  Finalmente, muestra el gráfico usando plt.show() para visualizar el mapa de calor en la pantalla.

2.2. Dispersión espacial de los precios

Para crear un gráfico de dispersión que muestre el valor de las casas en función de su ubicación, sigue estos pasos:

- **Configurar el tamaño del gráfico**:

- Usa plt.figure(figsize=(ancho, alto)) de la librería matplotlib para definir el tamaño del gráfico.

- **Crear el gráfico de dispersión**:
  - Emplea sns.scatterplot() del paquete seaborn para generar el gráfico. Especifica el DataFrame que contiene los datos, así como las variables Longitude y Latitude para los ejes X e Y, respectivamente. Utiliza el parámetro hue para colorear los puntos según el Median_House_Value y elige una paleta de colores con el parámetro palette.

- **Añadir la barra de color**:
  - Utiliza plt.Normalize() para normalizar los valores del Median_House_Value, lo que te ayudará a mapear los colores adecuadamente.
  - Crea un objeto ScalarMappable con plt.cm.ScalarMappable() para generar la barra de color, utilizando la misma paleta de colores y normalización.
  - Usa plt.colorbar() para agregar la barra de color al gráfico, y no olvides etiquetarla con un título que indique que representa el "Valor de la Casa Mediana".

- **Añadir un título**:
  - Emplea plt.title() para proporcionar un título descriptivo al gráfico que explique lo que se está mostrando.

- **Mostrar el gráfico**:
  - Finalmente, utiliza plt.show() para visualizar el gráfico de dispersión en la pantalla.

Este gráfico te permitirá analizar la relación entre la ubicación de las casas y su valor, ayudándote a identificar patrones y tendencias en los datos.

2.3. Visualización de la Densidad de Valores de Casas por Ubicación

Este gráfico hexbin muestra la relación entre la longitud y la latitud de las casas en California, representando el promedio del valor mediano de las

casas en cada hexágono. Utilizando hexágonos para agrupar los datos, el gráfico proporciona una visualización más clara de la densidad de valores en áreas específicas.

Para crear un gráfico de dispersión que muestre el valor de las casas en función de su ubicación, sigue estos pasos:

1. Configurar el tamaño del gráfico

- Usa `plt.figure(figsize=(ancho, alto))` de la librería `matplotlib` para definir el tamaño del gráfico.

2. Crear el gráfico de dispersión

- Emplea `sns.scatterplot()` del paquete `seaborn` para generar el gráfico. Especifica el DataFrame que contiene los datos, así como las variables `Longitude` y `Latitude` para los ejes X e Y, respectivamente. Utiliza el parámetro `hue` para colorear los puntos según el `Median_House_Value` y elige una paleta de colores con el parámetro `palette`.

3. Añadir la barra de color

- Utiliza `plt.Normalize()` para normalizar los valores del `Median_House_Value`, lo que te ayudará a mapear los colores adecuadamente.

- Crea un objeto `ScalarMappable` con `plt.cm.ScalarMappable()` para generar la barra de color, utilizando la misma paleta de colores y normalización.

- Usa `plt.colorbar()` para agregar la barra de color al gráfico, y no olvides etiquetarla con un título que indique que representa el "Valor de la Casa Mediana".

4. Añadir un título

- Emplea `plt.title()` para proporcionar un título descriptivo al gráfico que explique lo que se está mostrando.

5. Mostrar el gráfico

- Finalmente, utiliza `plt.show()` para visualizar el gráfico de dispersión en la pantalla.

Este gráfico te permitirá analizar la relación entre la ubicación de las casas y su valor, ayudándote a identificar patrones y tendencias en los datos.

3. Preprocesamiento de Datos

3.1 Evaluando Outliers

Para analizar los valores atípicos en el conjunto de datos, procederemos de la siguiente manera:

Establecemos un diccionario llamado limites, donde cada columna del DataFrame se asocia al valor 0.98. Esto indica que estamos interesados en observar el comportamiento de las distribuciones al excluir el 2% de los valores más altos.

```
limites = {columna: 0.98 for columna in data.columns}
```

Para visualizar comparativamente los histogramas con y sin el 2% de datos más grandes procedemos con la **creación de la Función clear_hist**:

- Definimos una función llamada clear_hist, que tomará como entrada el DataFrame y el nombre de la columna que queremos evaluar.
- Dentro de la función:
    - Imprimimos el nombre de la columna que estamos evaluando.
    - Calculamos el límite del 98% para la columna utilizando el método .quantile().
    - Creamos una figura con dos subgráficas:
        - La primera subgráfica mostrará el histograma de la distribución original de la variable.
        - La segunda subgráfica mostrará el histograma de la misma variable, pero excluyendo los valores que superan el límite calculado.

- Finalmente, mostramos ambos histogramas para una comparación visual.

Ahora que tenemos la función que nos ayudara a comparar los histogramas la aplicamos sobre cada columna del dataset por medio de un bucle.

```
for x in data.columns:
  clear_hist(data,x)
```

3.2 Truncando la variable

Ahora que hemos identificado las columnas que requieren truncamiento al 2%, procederemos a modificar el diccionario de límites y a truncar los valores de las variables según su respectivo límite.

- **Bucle a través de las columnas**:

- Usamos un bucle for para iterar sobre cada par de elementos en el diccionario limites, donde var representa el nombre de la columna y limit es el valor que hemos asignado (0.98 en este caso).

- **Calcular el valor mínimo**:

- min = data[var].min(): Esta línea calcula y almacena el valor mínimo de la columna actual (var) en la variable min. Este valor se utilizará como el límite inferior al truncar los datos.

- **Calcular el límite superior**:

- limit_ = data[var].quantile(limit): Aquí se utiliza el método .quantile() para calcular el límite superior para la columna actual. Este límite representa el valor correspondiente al percentil 98 de la distribución de la variable, es decir, el valor por encima del cual se consideran los 2% más altos.

- **Truncar los valores de la columna**:

- data[var] = data[var].clip(min, limit_): La función .clip() se utiliza para truncar los valores de la columna actual. Esta función reemplaza todos los valores menores que el mínimo (que no se modifican) y todos los valores mayores que el límite superior (limit_) con el valor de limit_. Esto significa que cualquier valor por encima del percentil 98 se ajustará a ese valor.

Al finalizar el bucle, habremos truncado los valores atípicos de cada columna que requiere truncamiento, limitando así los efectos de estos valores extremos en el análisis posterior. Esto ayudará a mejorar la calidad de los modelos y las visualizaciones, permitiendo un análisis más robusto de los datos.

Ahora que tenemos los datos truncados, visualizamos los nuevos histogramas.

```
for x in vars:
  sns.histplot(data,x=x,bins=10)
  plt.show()
```

4. **Construcción de Modelos**

Primero, se importan las librerías necesarias: xgboost, train_test_split para dividir los datos, y mean_squared_error para evaluar el modelo. Luego, se crea el DataFrame X eliminando la columna Median_House_Value, que contendrá todas las características a usar para las predicciones. La variable y se define como Median_House_Value, que es la variable que queremos predecir.

Con X y y listos, se pueden dividir los datos en conjuntos de entrenamiento y prueba, lo cual es fundamental para entrenar y evaluar el modelo.

```
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
X = data.drop('Median_House_Value', axis=1)
y = data['Median_House_Value']
```

Partición de datos

train_test_split: Esta función de sklearn.model_selection permite dividir un conjunto de datos en dos partes: un conjunto de entrenamiento (para entrenar el modelo) y un conjunto de prueba (para evaluar su rendimiento).

X y y: Estas son las características (X) y la variable objetivo (y) que hemos preparado previamente.

test_size=0.2: Este argumento especifica que el 20% de los datos se utilizará para el conjunto de prueba, mientras que el 80% restante se destinará al conjunto de entrenamiento. Esto es un enfoque común para asegurar que haya suficientes datos para entrenar el modelo, pero también suficientes para evaluar su rendimiento.

random_state=42: Este parámetro establece una semilla para el generador de números aleatorios, lo que garantiza que la división de los datos sea reproducible. Es decir, si ejecutas el código varias veces, obtendrás la misma división de datos, lo cual es útil para la consistencia en los experimentos.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

4.1 xgboost

Aquí se está creando y entrenando un modelo de regresión usando XGBoost:

Crear el modelo: Se inicializa el modelo XGBRegressor con parámetros que definen su comportamiento, como la función de pérdida (error cuadrático), el número de árboles (100) y la tasa de aprendizaje (0.1).

Entrenar el modelo: Se utiliza el método .fit() para entrenar el modelo con los datos de entrenamiento (X_train y y_train). Esto permite que el modelo aprenda la relación entre las características y el valor que se desea predecir.

Al finalizar, tendrás un modelo listo para hacer predicciones.

```
model = xgb.XGBRegressor(objective='reg:squarederror', n_estimators=100, learning_rate=0.1)
model.fit(X_train, y_train)
```

4.2 lightgbm

Se está creando y entrenando un modelo de regresión utilizando LightGBM:

Crear el modelo: Se inicializa el modelo LGBMRegressor con parámetros como la función objetivo ('regression'), que indica que es un modelo de regresión, el número de árboles a construir (n_estimators=100), y la tasa de aprendizaje (learning_rate=0.1).

Entrenar el modelo: Se utiliza el método .fit() para entrenar el modelo con los datos de entrenamiento (X_train y y_train). Durante este proceso, el

modelo aprende a predecir el valor objetivo en función de las características proporcionadas.

Al completar este paso, tendrás un modelo de LightGBM listo para realizar predicciones.

```
import lightgbm as lgb
model = lgb.LGBMRegressor(objective='regression', n_estimators=100, learning_rate=0.1)
model.fit(X_train, y_train)
```

5. **Evaluación y Selección del Modelo**

Para evaluar y comparar el rendimiento de los modelos XGBoost y LightGBM, puedes seguir estos pasos:

Importar las Métricas: Utiliza funciones de sklearn.metrics para calcular las métricas de error y precisión. Esto incluye:

Error Cuadrático Medio (RMSE): Mide la raíz cuadrada del error cuadrático medio. Indica la magnitud promedio de los errores en las predicciones.

Error Absoluto Medio (MAE): Mide el promedio de las diferencias absolutas entre las predicciones y los valores reales. Proporciona una medida sencilla de error.

R^2 (Coeficiente de Determinación): Indica qué proporción de la variabilidad en la variable objetivo es explicada por el modelo. Un valor más cercano a 1 indica un mejor ajuste.

Error Porcentual Absoluto Medio (MAPE): Mide el error en porcentaje, lo que permite una comparación fácil entre diferentes escalas de datos.

Hacer Predicciones:

Para cada modelo (XGBoost y LightGBM), utiliza el método .predict() con el conjunto de prueba (X_test) para obtener las predicciones.

Almacena estas predicciones en variables separadas (y_pred_xgb para XGBoost y y_pred_lgb para LightGBM).

Calcular las Métricas:

Utiliza las predicciones y los valores reales (y_test) para calcular las métricas de error:

Para XGBoost:

Calcula rmse_xgb, mae_xgb, r2_xgb, y mape_xgb usando las funciones importadas.

Para LightGBM:

Calcula rmse_lgb, mae_lgb, r2_lgb, y mape_lgb de la misma manera.

Finalmente compara estos indicadores para determinar cual es el mejor modelo.