Training

December 18, 2024

Importamos las librerias necesarias

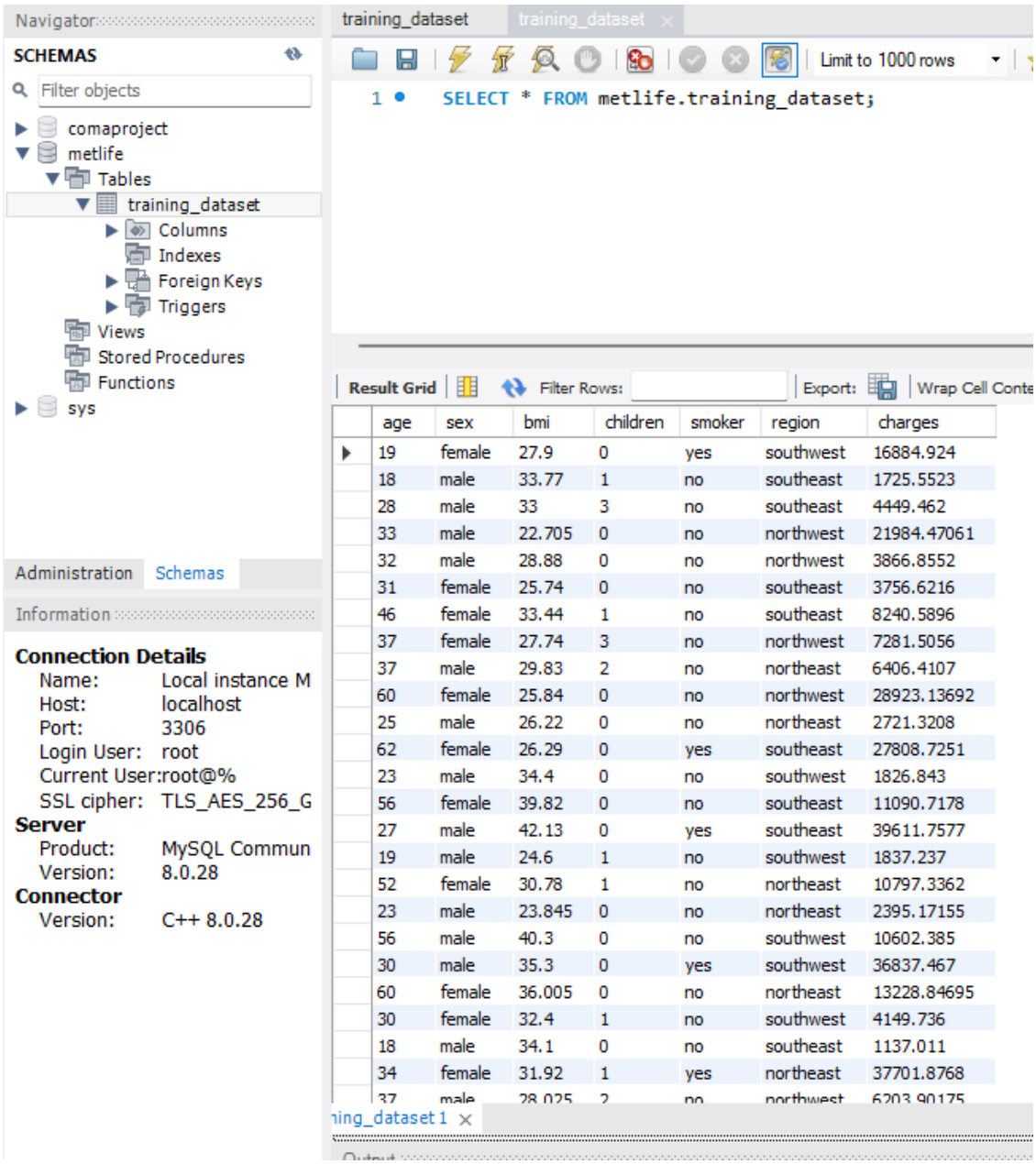
[85]:
```
import pymysql
import warnings
warnings.filterwarnings('ignore')
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from scipy.stats import zscore
from pyod.models.mad import MAD
from scipy.stats import pearsonr
from sklearn.model_selection import train_test_split
from scipy import stats
from scipy.stats import zscore
import joblib
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.linear_model import LinearRegression
```

Realizamos la conexión a la base de datos de MySQL, previamente creada (usando el archivo csv compartido). Al mismo tiempo que, creamos una excepción en caso de que se llegué a generar un mensaje de error.

[86]:
```
db_name = "METLIFE"
db_host = "localhost"
db_username = "root"
db_password = "123456"

try:
    conexion = pymysql.connect(host = db_host,
                               port = int(3306),
                               user = "root",
                               password = db_password,
                               db = db_name)
except e:
    print (e)
```

```
if conexion:
    print ("Conexión exitosa")
else:
        print ("error")
```

```
Conexión exitosa
```



Una vez realizada la conexión, obtenemos todo el contenido de la tabla mediente una sentencia de SQL y almacenamos el contenido en la variable df.

[87]:
```
df = pd.read_sql_query("select * from training_dataset", conexion)
```

Como parte del primer acercamiento hacia la data, buscamos identificar valores nulos dentro de la tabla que pudierán llegar a afectar el rendimiento del modelo. Especialmente en los siguientes aspectos: Sesgos: Si los valores nulos no se manejan adecuadamente, podrían introducir sesgos en el modelo.

Pérdida de precisión: Las columnas o filas con muchos valores nulos pueden aportar poca información al modelo, reduciendo la capacidad predictiva.

[88]:
```
null_counts = df.isnull().sum()
print(null_counts)
```

```
age         0
sex         0
bmi         0
children    0
smoker      0
region      0
charges     0
dtype: int64
```

En este caso, vemos que no se cuentan con valores nulos por lo que no es necesario realizar alguna imputacion de datos.

De igual forma, obtenemos el tipo de dato por columna. Esto para que ver que en la lectura se haya identificado el tipo de dato correcto.

[89]:
```
print(df.dtypes)
```

```
age           int64
sex          object
bmi         float64
children      int64
smoker       object
region       object
charges     float64
dtype: object
```

Como segundo punto del analisis que buscamos realizar hacia la data. Usamos la función describe() esto, para identificar la distribución de la data y las magnitudes de cada columna.

[90]:
```
print(df[['age', 'bmi', 'children', 'charges']].describe())
```

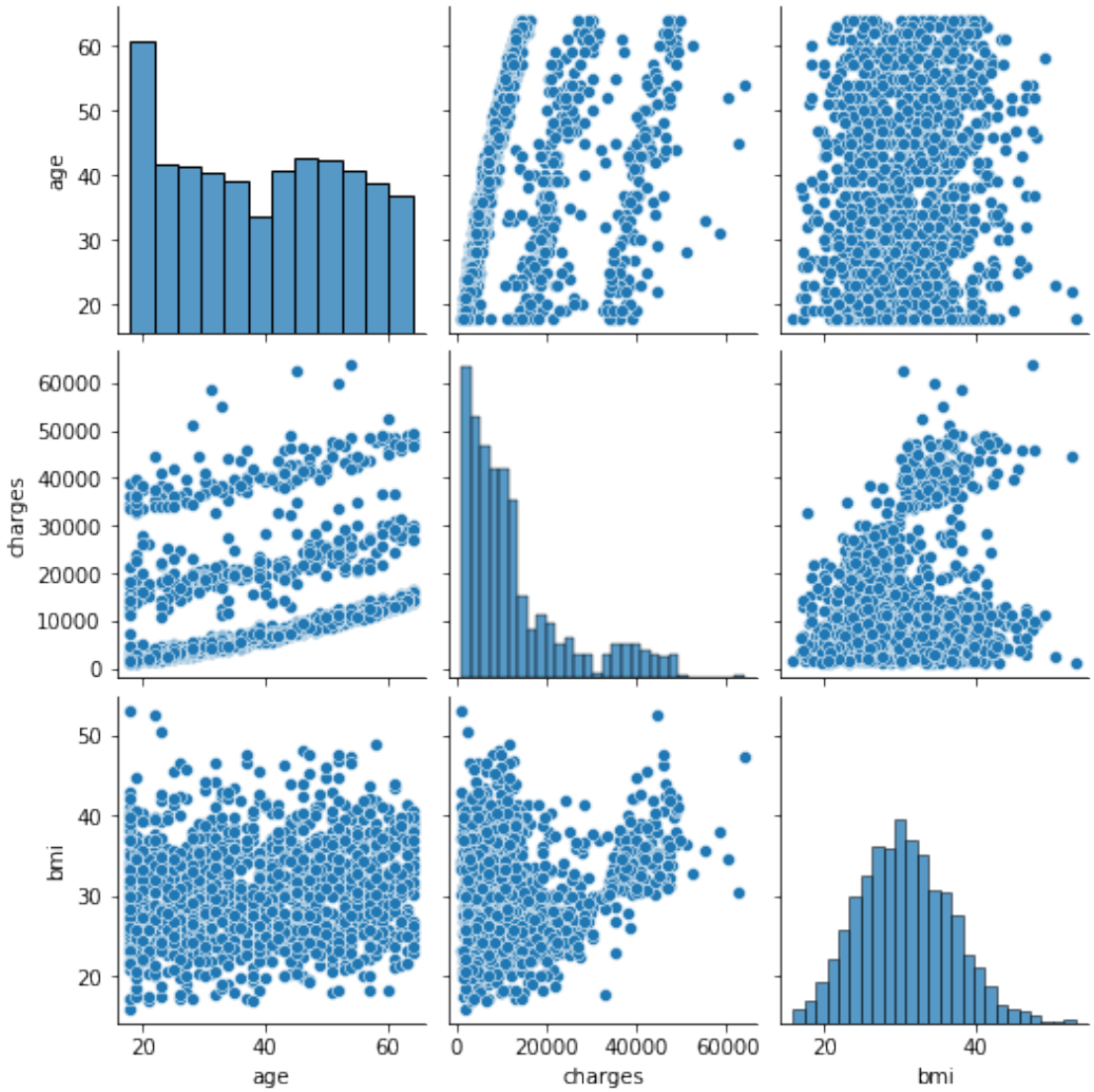
```
                age          bmi     children       charges
count  1338.000000  1338.000000  1338.000000   1338.000000
mean     39.207025    30.663397     1.094918  13270.422265
std      14.049960     6.098187     1.205493  12110.011237
min      18.000000    15.960000     0.000000   1121.873900
25%      27.000000    26.296250     0.000000   4740.287150
50%      39.000000    30.400000     1.000000   9382.033000
```

```
75%       51.000000     34.693750      2.000000   16639.912515
max       64.000000     53.130000      5.000000   63770.428010
```

Logramos identificar que: - La columna “charges” tiene una diferencia en maginitudes en relación con las demás variables, esto puede ocasionar un mal rendimiento y afectar la precisión del modelo. Por otro lado, después de observar los cuantiles vemos que el 75% de las personas tienen un costo médico menor o igual a 16,639.91 y un precio promedio de 13, 270.422265 y valor máximo de 63,770.428010, esto nos dice que la distribución se encuentra sesgada hacia la izquierda.

- La columna “bmi” presenta una distribución normal ya que la media de valores se encuentran en torno al 30.663397, un valor minimo de 15.960000 y un valor máximo de 53.130000
- Con la columna “children” vemos que, el 75% de las personas tienen 2 hijos o menos.

[91]:
```
# Especificamos las columnas que quiero incluir en el pairplot
columnas_interes = ["age", "charges", "bmi"]

# Creamos el pairplot para las columnas seleccionadas
sns.pairplot(data=df, vars=columnas_interes)

# Mostramos el gráfico
plt.show()
```



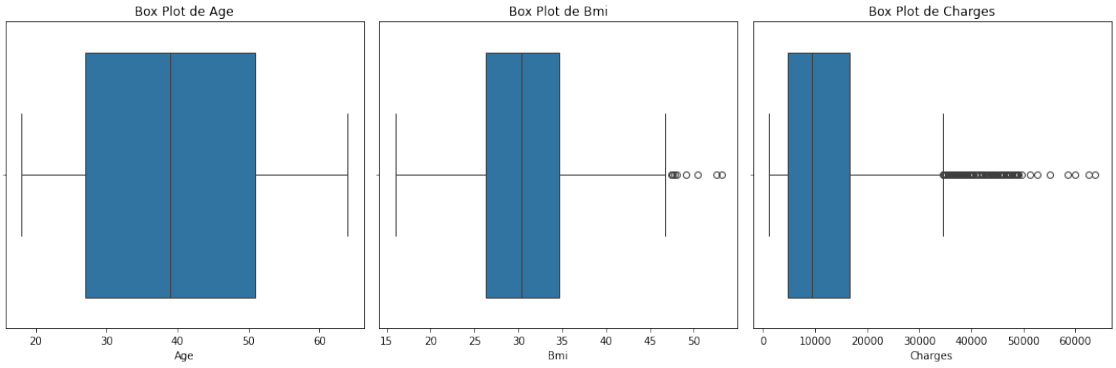
De manera visual, podemos ver que: - Entre las variables “age” y “charges” existe una relación positiva: a medida que la edad (“age”) aumenta, los cargos (“charges”) tienden a aumentar también. - Entre las variables “bmi” y “charges” la relación parece no lineal: A medida que el índice de masa corporal (bmi) aumenta, también lo hacen los cargos (charges), pero la dispersión es alta. También podemos ver que, hay casos particulares de personas con charges muy elevados, lo que sugiere que podría haber otros factores, como condiciones de salud o el hábito de fumar, que influyen en esta relación. - La distribución de “charges” sugiere la posibilidad de realizar transformaciones como el logaritmo para estabilizar la varianza.

[92]:
```
df['log_charges'] = np.log(df['charges'] )
```

Realizamos la transformación de la variable “charges” aplicando la función de logartimo y almacenando los nuevos valores en una columna llamada “log_charges”.

Creamos BOXPLOT para las columnas "age", "bmi" y "charges" y de esa forma identificar de manera visual tanto la distribución de los datos como los valores "outliers".

[93]:
```
# Creamos una figura con subgráficos
fig, axes = plt.subplots(1, 3, figsize=(15, 5), sharey=False)

# Variables a graficar
variables = ['age', 'bmi', 'charges']

# Graficamos cada variable en un subgráfico
for ax, var in zip(axes, variables):
    g = sns.boxplot(data=df, x=var, ax=ax)
    g.set_title(f'Box Plot de {var.capitalize()}')
    g.set_xlabel(f'{var.capitalize()}')

# Ajustamos el espaciado
plt.tight_layout()
plt.show()
```



Logramos identificar que:

Box Plot de Age (Edad): La distribución parece ser bastante simétrica. No se observan outliers (valores atípicos) significativos, ya que no hay puntos dispersos fuera de los bigotes. La mayoría de los datos de edad se concentran entre aproximadamente 20 y 60 años.

Box Plot de Bmi (Índice de Masa Corporal): Aquí sí se observan outliers a la derecha del diagrama, lo que sugiere que existen valores altos de Bmi que son atípicos en comparación con el resto de los datos. El rango intercuartílico (IQR) está entre aproximadamente 25 y 35, donde se concentra la mayoría de los datos.

Box Plot de Charges (Costos o Cargos): En este caso, hay una gran cantidad de outliers en la parte superior del diagrama, lo que indica que existen valores extremadamente altos de Charges en comparación con el resto de la muestra. Esto sugiere una distribución sesgada a la derecha, ya que la mayoría de los datos se concentran en valores bajos y hay una cola larga hacia valores más altos.

Mostramos los valores únicos y los conteos para cada columna categorica, esto nos ayudará a:
-Detectar datos inconsistentes o errores.

-Entender la distribución de las categorías.

[94]:
```
columns_to_check = ['sex', 'children', 'smoker', 'region']

# Mostramos valores únicos y sus conteos para cada columna seleccionada
for column in columns_to_check:
    print(f"Valores únicos y conteos en la columna '{column}':")
    print(df[column].value_counts())
    print()
```

```
Valores únicos y conteos en la columna 'sex':
sex
male      676
female    662
Name: count, dtype: int64

Valores únicos y conteos en la columna 'children':
children
0    574
1    324
2    240
3    157
4     25
5     18
Name: count, dtype: int64

Valores únicos y conteos en la columna 'smoker':
smoker
no     1064
yes     274
Name: count, dtype: int64

Valores únicos y conteos en la columna 'region':
region
southeast    364
southwest    325
northwest    325
northeast    324
Name: count, dtype: int64
```

Vemos que, no existen datos inconsistes o errores dentro de las variables categoricas. Vale la pena mencionar que, los "outliers" como tal no se aplican a las variables categóricas en el mismo sentido que a las variables numéricas ya que en variables categoricas unicamente se logran identificar frecuencias.

0.0.1 Extracción de valores "outliers"

Z-Score Un Z-Score, también conocido como puntuación Z, es una medida estadística que indica cuántas desviaciones estándar un punto de datos específico está por encima o por debajo de la media del conjunto de datos.

En este sentido, podemos calcular la puntuación Z de cada dato usando la función Z-Score de scipy y compararla con un umbral para determinar qué valores son considerados atípicos. Por lo general se establece un umbral de 3, por lo que aquellos puntos de datos cuya puntuación Z absoluta sea superior a 3 son outliers.

Recordando que: Z-Score sólo es apropiada para distribuciones normales

[95]:
```
# Calculamos el z-score para la columna 'bmi'

z_scores = zscore(df['bmi'])
abs_z_scores = np.abs(z_scores)

# Creamos una nueva columna llamada 'outliers' que nos ayudará a marcar si una␣
 ↪fila es un outlier.
df['outliers'] = abs_z_scores > 3


print(f'Número de outliers totales para la columna bmi: {df["outliers"].sum()}')
```

```
Número de outliers totales para la columna bmi: 4
```

Z-Score modificado Cuando los datos son asimétricos o no se distribuyen de forma normal podemos utilizar el Z-score modificado, también conocido como MAD-Z-Score. Este, a diferencia del z-score, utiliza la mediana y la desviación absoluta mediana (MAD en inglés) en lugar de la media y la desviación estándar con el fin de evitar el efecto de los outliers sobre estas dos últimas medidas. Se recomienda que los valores con puntuaciones z modificadas inferiores a -3,5 o superiores a 3,5 se etiqueten como posibles valores atípicos.

[96]:
```
# Configuramos MAD con un threshold de 3.5
mad = MAD(threshold=3.5)

# Analizamos los outliers en la columna 'charges'
charges_reshaped = df['charges'].values.reshape(-1, 1)
charges_labels = mad.fit(charges_reshaped).labels_

# Analizamos los outliers en la columna 'age'
age_reshaped = df['age'].values.reshape(-1, 1)
age_labels = mad.fit(age_reshaped).labels_

# Actualizamos la columna 'outliers' en el DataFrame ahora usando las columnas␣
 ↪"age" y "charges"
df['outliers'] = np.where(
    (df['outliers'] == True) | (charges_labels == 1) | (age_labels == 1),
```

```
    True,
    False
)

# Imprimimos el número de outliers y el DataFrame con la columna actualizada
print(f'Número de outliers totales (incluyendo los valores de la columna "bmi"):␣
 ↪{df["outliers"].sum()}')
```

```
Número de outliers totales (incluyendo los valores de la columna "bmi"): 133
```

En este caso, imprimimos la suma de los valores de la columna "outliers". Esta columna nos ayuda a identificar los valores que se clasifican como "outliers" para todas las columnas numericas ("bmi", "age" y "charges").

[97]:
```
# Sobreescribimos el DataFrame original eliminando las filas con outliers
df = df[df['outliers'] == False].copy()

# Imprimimos el número de filas restantes. despues de eliminar los "outliers"
print(f'Número de filas restantes después de eliminar los "outliers": {len(df)}')

# Eliminamos la columna 'outliers'
df.drop(columns=['outliers'], inplace=True)
```

```
Número de filas restantes después de eliminar los "outliers": 1205
```

Después de, identificar los valores "outliers" dentro del dataframe para las variables numericas eliminamos los registros para: - Eliminar sesgos en los resultados del modelo. - Evitar reducir la precisión y generalización del modelo porque los algoritmos intentan ajustarse a estos puntos anormales, en lugar de centrarse en el patrón general de los datos. - Evitar la interpretación erronea ya que los outliers pueden dificultar la interpretación de patrones y relaciones entre las variables.

Codificación de variables categóricas Las variables categóricas representan atributos discretos que no se pueden interpretar como valores numéricos directos. Al codificar estas variables de manera adecuada, podemos transformarlas en formatos numéricos comprensibles para los algoritmos de modelado.

Por lo que, usaremos la codifcación ordinal para asignar un valor numerico a cada valor que puede tomar una columna categorica.

[98]:
```
# Definimos el orden de las categorías para las columnas
encoding_orders = {
    'sex': ['male', 'female'],
    'smoker': ['no', 'yes'],
    'region': ['southeast','southwest','northwest','northeast']
}

# Aplicamos la codificación ordinal para cada columna
for column, order in encoding_orders.items():
```

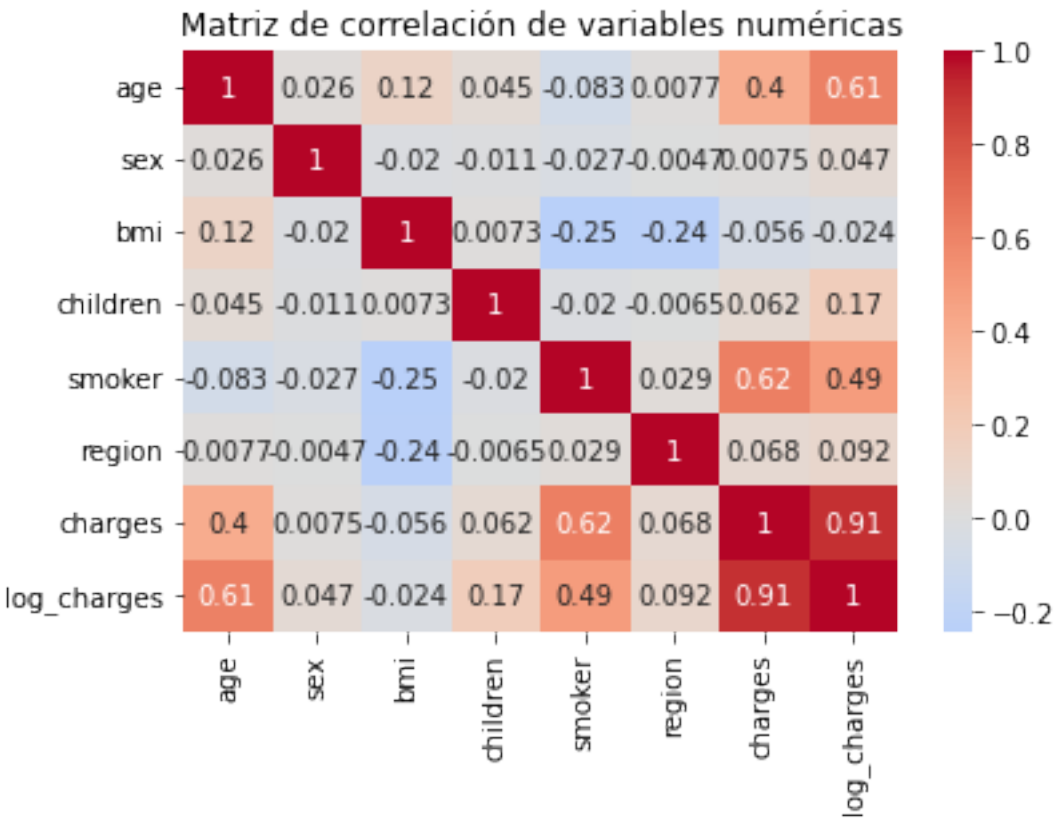
```
    df[column] = df[column].map({cat: idx for idx, cat in enumerate(order)})

# Imprimimos el DataFrame codificado
print(df)
```

```
      age  sex     bmi  children  smoker  region      charges  log_charges
0      19    1  27.900         0       1       1  16884.92400     9.734176
1      18    0  33.770         1       0       0   1725.55230     7.453302
2      28    0  33.000         3       0       0   4449.46200     8.400538
3      33    0  22.705         0       0       2  21984.47061     9.998092
4      32    0  28.880         0       0       2   3866.85520     8.260197
...   ...  ...     ...       ...     ...     ...          ...          ...
1333   50    0  30.970         3       0       2  10600.54830     9.268661
1334   18    1  31.920         0       0       3   2205.98080     7.698927
1335   18    1  36.850         0       0       0   1629.83350     7.396233
1336   21    1  25.800         0       0       1   2007.94500     7.604867
1337   61    1  29.070         0       1       2  29141.36030    10.279914

[1205 rows x 8 columns]
```

Despues de, identificar las frecuencias para cada columna categorica es facil definir los valores que puede tomar cada columna.

Matriz de correlación Creamos una matriz de correalación para entender las relaciones entre las variables en el conjunto de datos. Además de, identificar variables relevantes y guiar en la mejor elección del modelo (lineal o no lineal), mejorando el rendimiento y la interpretabilidad del modelo.

[99]:
```
correlation_matrix = df.select_dtypes(include=['number']).corr()

# Creamos un mapa de calor para visualizar la matriz de correlación
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", center=0)
plt.title('Matriz de correlación de variables numéricas')
plt.show()
```



-Log_Charges y Smoker: Existe una correlación moderada positiva de 0.49 entre smoker y log_charges. Esto indica que las personas que fuman tienden a tener costos (charges) más altos. Además de que al aplicar el logaritmo a charges, notamos una reducción en la correlación ya que estamos comprimendo la escala de valores de charges y reduciendo la influencia de los valores extremos. Como vemos, esto disminuye la magnitud de la correlación, ya que los valores altos de charges (que estaban fuertemente asociados con smoker) tienen menos peso después de la transformación.

-Log_Charges y BMI: El valor -0.024 está muy cercano a 0, lo que indica que no hay una relación lineal apreciable entre las dos variables. Aplicar una transformación logarítmica en bmi no tendría un efecto significativo porque el problema no es la asimetría de la distribución, sino que bmi no está relacionado con log_charges ya sea de manera lineal o no lineal.

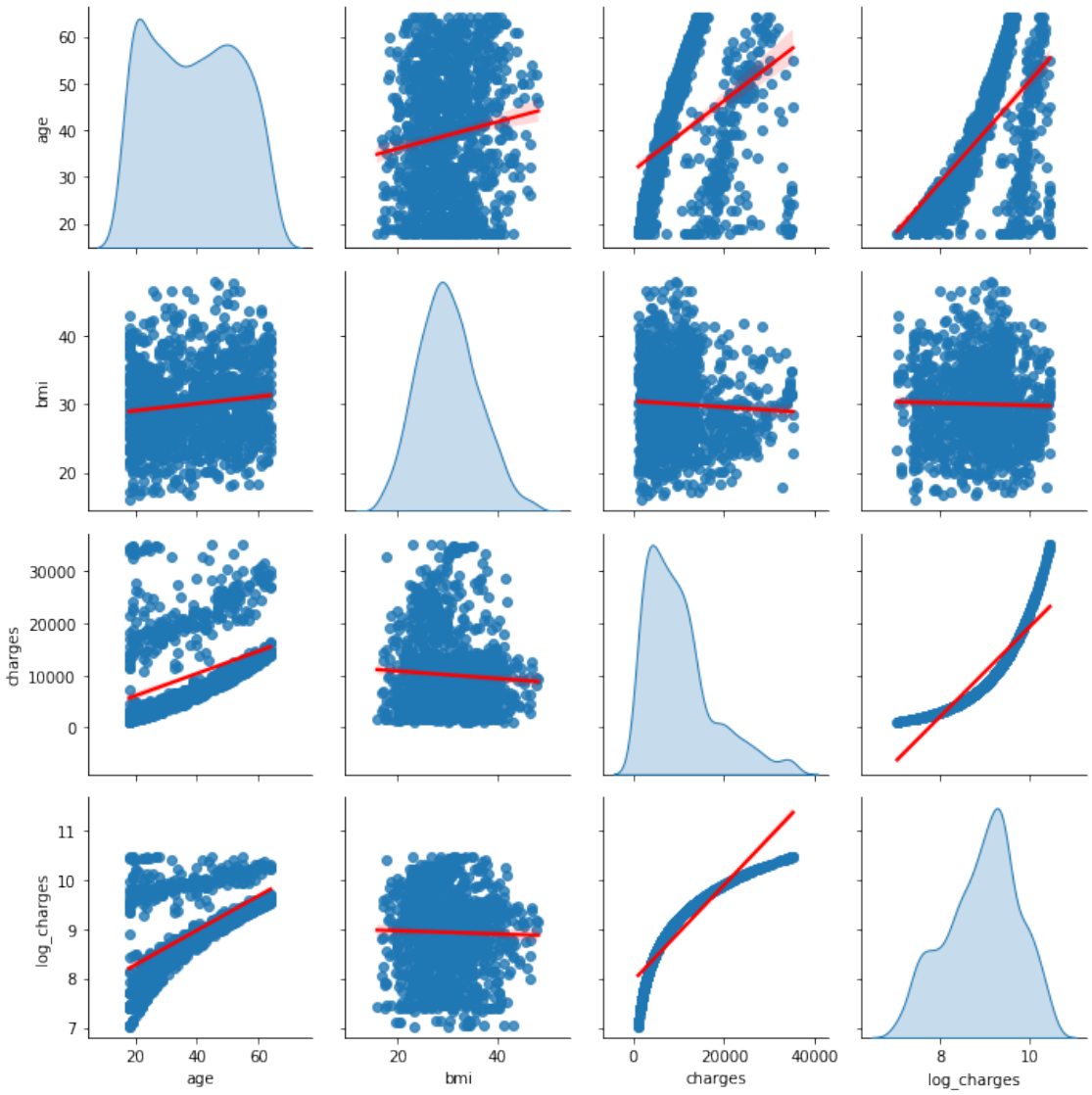
-Log_Charges y Age: La correlación es positiva moderada-alta (0.61), lo que indica que las personas de mayor edad tienden a tener mayores costos. Además de que, la correlación aumenta significativamente a 0.61 en comparación con charges (0.4), indicando que la transformación logarítmica ayuda a capturar mejor la relación con la edad.

-Relación con children, sex y region: Las variables children, sex y region muestran correlaciones muy bajas con charges y log_charges, lo que sugiere que no tienen un impacto significativo en

estas variables.

Resumen general: Edad y fumador son las variables que más influyen en los costos (charges), con age siendo el más relevante. La transformación logarítmica de charges mejora su relación con variables como age. Variables como children, sex y region no muestran correlaciones importantes con charges.

[100]:
```
columnas = ['age', 'bmi', 'charges','log_charges']

sns.pairplot(
    df[columnas], kind='reg', diag_kind='kde', plot_kws={'line_kws': {'color':␣
 ↪'red'}}
)
plt.tight_layout()
plt.show()
```



La línea roja representa una regresión ajustada para identificar tendencias entre las variables.

Variables involucradas:

- age (edad)
- bmi (índice de masa corporal)
- charges (cargos o costos)
- log_charges (logaritmo natural de los cargos)

Observaciones generales:

- Relación entre charges y age: Existe una relación positiva: a medida que aumenta la edad, los cargos (charges) tienden a incrementarse.
- Relación entre BMI y charges: No hay una relación fuerte aparente entre el índice de masa corporal (bmi) y los cargos (charges). La nube de puntos es dispersa y la línea roja es relativamente plana. Lo cual indica que BMI tiene un impacto limitado en los costos.

Distribuciones:

- Age presenta dos picos, lo que sugiere una distribución bimodal.
- Charges está sesgada a la derecha, lo que indica que la mayoría de los cargos son bajos, pero hay valores muy grandes (outliers).
- log_charges parece más simétrica, lo cual confirma que la transformación logarítmica logra manejar sesgos en los datos.

Conclusión: Este análisis sugiere que la edad (age) tiene una fuerte relación positiva con los costos (charges), mientras que otras variables como BMI no muestran una relación clara. La transformación logarítmica de charges es útil para reducir el sesgo en la distribución.

0.0.2 Regresión Polinomial

1. Relación no lineal entre algunas variables y charges En el análisis que realizamos: Observamos que age tenía una correlación moderada (0.4) con charges, pero al aplicar el logaritmo, mejoró a 0.61. La regresión polinomial puede capturar relaciones no lineales más complejas, en este caso, los costos médicos (charges) aumentan con la edad, pero no de manera lineal (relación cuadrática). Justificación: La regresión polinomial permite modelar estas relaciones curvilíneas que no pueden ser capturadas por un modelo lineal simple.

2. Evidencia de no linealidad en los gráficos En los gráficos de dispersión que vimos arriba: La relación entre age y charges parecía mostrar curvatura (un patrón no completamente lineal). La variable bmi también podría tener patrones más complejos, especialmente en combinación con otras variables como smoker.

Justificación: Esto nos dice que, un modelo lineal simple podría ser insuficiente para capturar la verdadera relación entre las variables y charges.La inclusión de términos polinómicos (como age^2 o bmi^2) puede ajustar mejor los patrones observados en los datos.

3. Interacción entre variables categóricas y continuas La variable smoker tiene una fuerte relación con charges (correlación de 0.49). Esto indica que ser fumador o no fumador influye significativamente en los costos. Sin embargo, el efecto de bmi o age podría depender de si la persona es fumadora o no (interacciones no lineales). Por ejemplo, el impacto del BMI en los costos podría ser más pronunciado para fumadores y menos relevante para no fumadores.

Justificación: Las interacciones no lineales y los efectos combinados de las variables categóricas (smoker, sex, region) con las continuas (age, bmi) pueden ser capturados mejor con términos polinómicos.

4. Mejora en las métricas del modelo Comparar un modelo lineal simple con un modelo polinomial: Calcular métricas como R^2 ajustado, RMSE (Error cuadrático medio) y MAE. Si el modelo polinomial reduce significativamente el error sin caer en sobreajuste, se justifica su uso. Realizar validación cruzada para asegurarse de que el modelo generaliza bien en datos nuevos.

Justificación: Si el modelo polinomial mejora las métricas de ajuste y precisión en comparación con el modelo lineal, esto respalda su uso.

5. Interpretación de las variables La regresión polinomial permite entender cómo las variables no lineales impactan en charges: Por ejemplo, un término age^2 podría indicar que los costos aumentan más rápidamente con la edad a partir de cierto punto. Un término bmi^2 podría mostrar que los costos solo se elevan significativamente cuando el BMI supera cierto umbral.

Justificación: La interpretación de los términos polinómicos ayuda a entender mejor el comportamiento no lineal de los datos y su efecto en charges.

Resumen: Argumento final El uso de un modelo de regresión polinomial para predecir charges se justifica porque:

La relación entre age y charges sugiere una posible curvatura (no linealidad). El impacto del BMI podría depender de umbrales críticos y combinaciones con otras variables, como smoker. La validación cruzada y la comparación de métricas de error podrían demostrar que el modelo polinomial ajusta mejor los datos. Se pueden capturar interacciones no lineales entre variables categóricas y continuas.

[112]:
```
X,y=df[["smoker","children","age",'region',"bmi","sex"]],df["log_charges"]
poly=PolynomialFeatures(degree=2,include_bias=False)
poly_features=poly.fit_transform(X)
X_train,X_test,y_train,y_test=train_test_split(poly_features,y,test_size=0.
↪3,random_state=43)
```

1.- Selección de variables: X, y = df[[“smoker”, “children”, “age”, ‘region’, “bmi”, “sex”]], df[“log_charges”]

X contiene las características predictoras seleccionadas (variables independientes) del DataFrame df, mientras que y es la variable objetivo log_charges (variable dependiente).

2.-Generación de características polinómicas: poly = PolynomialFeatures(degree=2, include_bias=False) poly_features = poly.fit_transform(X)

- Se utiliza PolynomialFeatures de sklearn para crear características polinómicas de grado 2. Esto significa que se incluirán términos polinómicos de todas las variables en X, pero sin incluir el término de sesgo (constante), porque include_bias=False.

3.- División del conjunto de datos: X_train, X_test, y_train, y_test = train_test_split(poly_features, y, test_size=0.3, random_state=43)

Se utiliza train_test_split para dividir los datos en entrenamiento (70%) y prueba (30%). El parámetro random_state=43 asegura reproducibilidad (semilla), es decir, la división será la misma cada vez que se ejecute el código.

[113]:
```
poly_reg_model=LinearRegression()
poly_reg_model.fit(X_train,y_train)
poly_y_predict=poly_reg_model.predict(X_test)
poly_rmse=np.sqrt(mean_squared_error(y_test,poly_y_predict))
print(f"poly rmse= {poly_rmse}")
```

```
poly rmse= 0.33580572114788526
```

1. Creación del modelo de regresión lineal: poly_reg_model = LinearRegression()

Se instancia un modelo de regresión lineal LinearRegression() de sklearn. Este modelo ajustará una línea recta (o hiperplano) a los datos, aunque en este caso las características polinómicas permitirán capturar relaciones no lineales.

2. Entrenamiento del modelo: poly_reg_model.fit(X_train, y_train)

El modelo se entrena utilizando los datos de entrenamiento X_train (características polinómicas) y y_train (valores reales de log_charges). La función fit() ajusta los coeficientes de la regresión lineal minimizando el error cuadrático.

3. Predicción en el conjunto de prueba: poly_y_predict = poly_reg_model.predict(X_test)

Una vez entrenado, el modelo predice los valores de log_charges para las características del conjunto de prueba X_test. poly_y_predict almacena las predicciones generadas.

4. Cálculo del error (RMSE): poly_rmse = np.sqrt(mean_squared_error(y_test, poly_y_predict))

Se calcula el error cuadrático medio (MSE) entre los valores reales y_test y las predicciones poly_y_predict usando mean_squared_error. El RMSE (Root Mean Squared Error) se obtiene tomando la raíz cuadrada del MSE. Este métrica mide la desviación promedio entre los valores reales y las predicciones.

[103]:
```
r2 = r2_score(y_test,poly_y_predict)
print(f"R-squared: {r2}")
```

```
R-squared: 0.8174276353945875
```

La función r2_score de sklearn calcula el coeficiente de determinación R_2, que evalúa la calidad del modelo ajustado comparando las predicciones con los valores reales. El valor R-squared: 0.8174276353945875 indica que el modelo explica aproximadamente el 81.7% de la variabilidad de

la variable objetivo (log_charges) a partir de las características polinómicas. Mientras más cerca esté el R_2 de 1, mejor será el ajuste del modelo. Un valor de 0.817 sugiere un buen desempeño, siempre recordando no sobreajustar el modelo.

Cross-Validation cross_val_score realiza la validación cruzada con 5 particiones (cv=5): Divide los datos en 5 subconjuntos (folds). El modelo se entrena en 4 subconjuntos y se evalúa en el restante, repitiendo este proceso 5 veces.

[104]:
```
for degree in range(1, 5):  # Probar con grados de 1 a 5
    poly = PolynomialFeatures(degree=degree, include_bias=False)
    X_poly = poly.fit_transform(X)
    poly_reg_model = LinearRegression()

    # Evaluar el modelo con cross-validation
    scores = cross_val_score(poly_reg_model, X_poly, y, cv=5,␣
 ↪scoring='neg_mean_squared_error')
    print(f"Grado {degree} - RMSE: {(-scores.mean())**0.5}")
```

```
Grado 1 - RMSE: 0.4395896316785941
Grado 2 - RMSE: 0.3886617903262824
Grado 3 - RMSE: 0.3938852768620326
Grado 4 - RMSE: 0.4180439791380783
```

Selección del grado óptimo del polinomio que equilibra el ajuste y evita el sobreajuste. 1.- Itera sobre diferentes grados del polinomio (1 a 4).

2.- Genera características polinómicas para cada grado.

3.- Entrena y evalúa el modelo usando validación cruzada para medir su desempeño con el RMSE promedio.

scoring='neg_mean_squared_error':La métrica utilizada es el MSE negativo (para que sea compatible con cross_val_score).

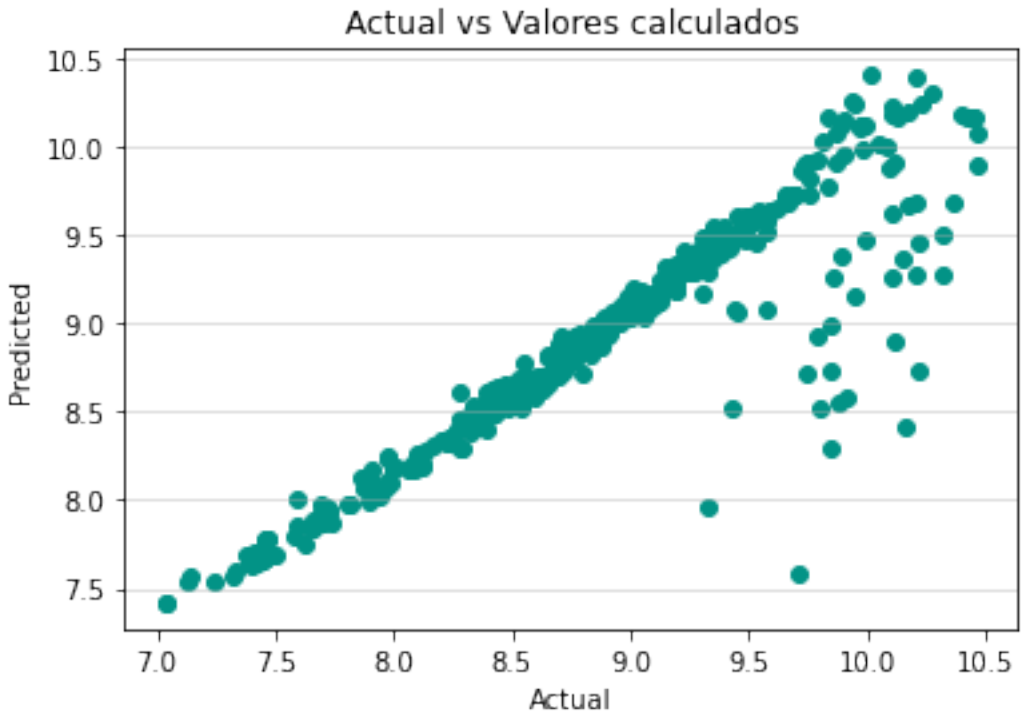
Sobreajuste (Overfitting): Para grados más altos (por ejemplo, grado 4 o 5), el modelo puede ajustarse demasiado a los datos de entrenamiento, lo que lleva a un sobreajuste.

Subajuste (Underfitting): Para grados más bajos, el modelo podría ser demasiado simple para capturar la relación compleja entre las variables, lo que también podría aumentar el RMSE promedio.

La idea es probar diferentes grados del polinomio y ver cuál proporciona el mejor rendimiento en promedio a lo largo de los pliegues de la validación cruzada. A pesar de que los valores de RMSE puedan variar entre los pliegues (y entre diferentes grados), lo que realmente nos interesa es saber cuál grado tiene el menor RMSE promedio a través de todos los pliegues.

[116]:
```
# Visualizamos la diferencia entre los precios actuales y los que el modelo␣
 ↪logró predecir
plt.scatter(y_test, poly_y_predict, color='#029386')
```

```
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('Actual vs Valores calculados')
plt.grid(axis='y', alpha=0.5)
plt.show()
```



Se genera un gráfico de dispersión que compara los valores reales (actuales) de la variable objetivo con las predicciones realizadas por el modelo.

1.- Definición de parametros para generar el grafico de dispersión y_test: Contiene los valores reales (actuales) de la variable objetivo.

poly_y_predict: Contiene las predicciones generadas por el modelo.

color='#029386': Define el color de los puntos en el gráfico.

2.- Etiquetas de los ejes: xlabel: Etiqueta el eje X como "Actual" (valores reales).

ylabel: Etiqueta el eje Y como "Predicted" (valores predichos).

Línea ideal: Si las predicciones fueran perfectas, todos los puntos deberían alinearse sobre una línea recta con pendiente 1 (línea Y = X).

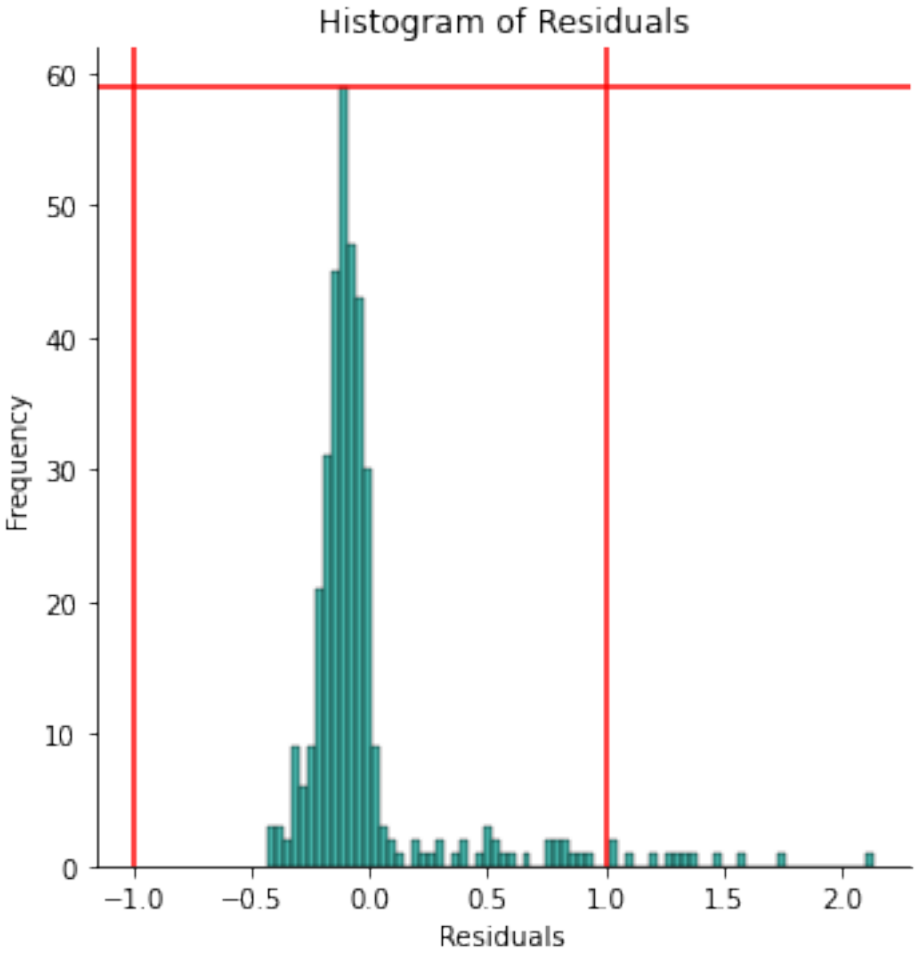
Comportamiento observado: La mayoría de los puntos están cercanos a la línea ideal, lo que indica que el modelo realiza predicciones precisas con la mayoria de los datos. Sin embargo, se

observa cierta dispersión en la parte superior derecha, lo que sugiere que el modelo tiene variaciones para los valores reales más altos.

Conclusión: El modelo tiene un buen desempeño con la mayoria de los datos, con predicciones ajustadas a los valores reales, pero existe un margen de error para valores extremos (posiblemente outliers).

[120]:
```
plt.figure(figsize=(12,8), dpi=1000)
sns.displot(y_test - poly_y_predict, color='#029386')
plt.title('Histogram of Residuals')
plt.xlabel('Residuals')
plt.ylabel('Frequency')

plt.axvline(-1, color='red', ymax=1)
plt.axvline(1, color='red', ymax=1)
plt.axhline(59, color='red', xmin=0, xmax=1)

plt.show()
```

```
<Figure size 12000x8000 with 0 Axes>
```



Forma de los residuos: La mayoría de los residuos están centrados cerca de 0, lo cual es deseable, ya que indica que las predicciones no están sistemáticamente sesgadas. Hay una distribución muy concentrada en torno a valores residuales pequeños.

Líneas rojas: Las líneas verticales en -1 y 1 destacan residuos que podrían considerarse outliers o valores atípicos. La línea horizontal en y = 59 marca un umbral de frecuencia máxima.

Comportamiento: El histograma sugiere que el modelo funciona bien para la mayoría de las predicciones, pero hay algunos valores extremos (residuos grandes) en los bordes de la distribución.

[114]:
```
# Guardamos el modelo
joblib.dump(poly_reg_model, 'modelo_entrenado.pkl')
```

[114]:
```
['modelo_entrenado.pkl']
```

[115]:
```
# Guardamos el transformador PolynomialFeatures
joblib.dump(poly, 'transformador_polynomial.pkl')
```

[115]:
```
['transformador_polynomial.pkl']
```