## SESIÓN REGRESIONES LINEALES

**CONTENIDOS:**

- Regresiones lineales simples: ¿qué es una regresión lineal simple?
- Determinación de los coeficientes de regresión.
- Métricas de error.
- Indicador de ajuste r2.
- Implementación en python con librería statsmodels.
- Regresiones lineales múltiples: qué es una regresión lineal múltiple.
- Supuestos de una regresión múltiple.
- Métodos de selección del modelo.
- Indicador r2-ajustado.
- Test de normalidad del error.
- Implementación en Python con librería statsmodels.

### REGRESIONES LINEALES SIMPLES

Las regresiones lineales son una de las técnicas más utilizadas en ciencia de datos y estadística para modelar la relación entre una variable dependiente y una o más variables independientes. Son ampliamente empleadas en la predicción y análisis de tendencias en diversos campos, como economía, finanzas, salud y marketing.

#### ¿Qué es una Regresión Lineal Simple?

Una regresión lineal simple modela la relación entre una variable dependiente (Y) y una variable independiente (X) mediante una línea recta. Su propósito es predecir el valor de Y en función de X, basándose en una relación lineal entre ambas variables.

Ecuación de la regresión lineal simple:

$$Y = \beta_0 + \beta_1 X + \epsilon$$

Donde:

- $\boldsymbol{\beta_0}$**:** Intercepto (valor de Y cuando X=0).
- $\boldsymbol{\beta_1}$ : Pendiente (cambio en Y por unidad de cambio en X).
- Ɛ: Termino de error.

El objetivo de la regresión es encontrar los valores óptimos de $\boldsymbol{\beta_0}$ y $\boldsymbol{\beta_1}$ que mejor ajusten la relación entre las variables.

**DETERMINACIÓN DE LOS COEFICIENTES DE REGRESIÓN.**

Para determinar los coeficientes de regresión ( $\boldsymbol{\beta_0}$ y $\boldsymbol{\beta_1}$ ), se utiliza el método de mínimos cuadrados, el cual minimiza la suma de los errores al cuadrado entre los valores observados y los predichos.

Fórmulas para calcular los coeficientes:

**Pendiente ($\boldsymbol{\beta_1}$ )**

$$\beta_1 = \frac{\sum(X_i - \bar{X})(Y_i - \bar{Y})}{\sum(X_i - \bar{X})^2}$$

Donde:

$X_i, Y_i$ son los valores individuales de las variables.

$\bar{X}, \bar{Y}$ son los valores promedio de X e Y.

**Intercepto**

$$\beta_0 = \bar{Y} - \beta_1 \bar{X}$$

El intercepto representa el valor esperado de *Y* cuando *X* es igual a cero.

## EJEMPLO DE REGRESIÓN LINEAL SIMPLE EN PYTHON

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Datos de ejemplo
X = np.array([1, 2, 3, 4, 5]).reshape(-1, 1)
Y = np.array([2, 3, 5, 6, 8])

# Crear modelo de regresión lineal
modelo = LinearRegression()
modelo.fit(X, Y)

# Obtener coeficientes
beta_0 = modelo.intercept_
beta_1 = modelo.coef_[0]

# Hacer predicciones
Y_pred = modelo.predict(X)

# Visualización de la regresión
plt.scatter(X, Y, color="blue", label="Datos reales")
plt.plot(X, Y_pred, color="red", label="Regresión Lineal")
plt.xlabel("Variable X")
plt.ylabel("Variable Y")
plt.title("Regresión Lineal Simple")
plt.legend()
plt.grid()
plt.show()

# Mostrar coeficientes
print(f"Intercepto (β0): {beta_0}")
print(f"Pendiente (β1): {beta_1}")
```

Este código implementa una Regresión Lineal Simple utilizando Python con la librería scikit-learn. Su objetivo es ajustar una línea recta a un conjunto de datos y hacer predicciones basadas en ella.

A continuación, se explica paso a paso lo que hace el código:

**Importación de librerías**

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
```

- NumPy (np): Se usa para manejar datos numéricos y manipular arreglos.
- Matplotlib (plt): Se emplea para visualizar los datos y la línea de regresión.
- LinearRegression de sklearn.linear_model: Se utiliza para crear y entrenar el modelo de regresión lineal.

**Creación de los Datos de Ejemplo**

```
# Datos de ejemplo
X = np.array([1, 2, 3, 4, 5]).reshape(-1, 1)
Y = np.array([2, 3, 5, 6, 8])
```

- X representa la variable independiente (por ejemplo, el número de horas estudiadas).
- Y representa la variable dependiente (por ejemplo, la calificación obtenida en un examen).
- reshape(-1,1) convierte X en un arreglo de dos dimensiones, necesario para sklearn.

**Creación y Entrenamiento del Modelo**

```
# Crear modelo de regresión lineal
modelo = LinearRegression()
modelo.fit(X, Y)
```

- Se crea un modelo de Regresión Lineal con LinearRegression().
- El modelo se entrena con modelo.fit(X, Y), ajustando la mejor línea recta a los datos.

**Obtención de los Coeficientes de la Regresión**

```
# Obtener coeficientes
beta_0 = modelo.intercept_
beta_1 = modelo.coef_[0]
```

- beta_0: Representa el intercepto, es decir, el valor de Y cuando X = 0.
- beta_1: Representa la pendiente, es decir, cuánto cambia Y por cada unidad de aumento en X.

Si la ecuación de la regresión es:

$$Y = \beta_0 + \beta_1 X$$

Estos coeficientes nos ayudan a interpretar la relación entre X y Y.

**Predicción de Valores con el Modelo**

```
# Hacer predicciones
Y_pred = modelo.predict(X)
```

- Se utilizan los valores de X para predecir los valores de Y con la ecuación de la regresión lineal.
- Y_pred contiene los valores ajustados por la línea de regresión.

**Visualización de la Regresión Lineal**

```
# Visualización de la regresión
plt.scatter(X, Y, color="blue", label="Datos reales")
plt.plot(X, Y_pred, color="red", label="Regresión Lineal")
plt.xlabel("Variable X")
plt.ylabel("Variable Y")
plt.title("Regresión Lineal Simple")
plt.legend()
plt.grid()
plt.show()
```

- plt.scatter(X, Y, color="blue"): Dibuja los puntos originales en color azul.
- plt.plot(X, Y_pred, color="red"): Dibuja la línea de regresión en color rojo.
- Se agregan etiquetas, título y una cuadrícula para mejorar la visualización.

**Impresión de los Coeficientes Calculados**

```
# Mostrar coeficientes
print(f"Intercepto (β0): {beta_0}")
print(f"Pendiente (β1): {beta_1}")
```

- Muestra los valores del intercepto y la pendiente, los cuales permiten predecir nuevos valores de Y en función de X.

**MÉTRICAS DE ERROR**

Las métricas de error son herramientas fundamentales para evaluar el desempeño de un modelo de Regresión Lineal. Nos permiten cuantificar qué tan lejos están las predicciones del modelo respecto a los valores reales.

**1. Error Cuadrático Medio (MSE - Mean Squared Error)**

El MSE es el promedio de los errores al cuadrado. Penaliza fuertemente los errores grandes, lo que lo hace útil cuando se busca minimizar desviaciones significativas en las predicciones.

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

Características:

- Penaliza errores grandes debido al cuadrado.
- Se expresa en unidades cuadradas, por lo que no es intuitivo interpretar su valor directamente.

**Ejemplo en Python:**

```python
from sklearn.metrics import mean_squared_error

# Valores reales y predicciones de ejemplo
Y_real = [2, 3, 5, 6, 8]
Y_pred = [2.2, 3.1, 4.8, 5.9, 7.5]

# Cálculo del MSE
mse = mean_squared_error(Y_real, Y_pred)
print(f"Error Cuadrático Medio (MSE): {mse}")
```

**2. Error Absoluto Medio (MAE - Mean Absolute Error)**

El MAE es el promedio de los errores absolutos entre los valores reales y las predicciones. Es más robusto ante valores atípicos que el MSE.

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^{n} |Y_i - \hat{Y}_i|$$

Características:

- Se mide en las mismas unidades que los datos originales.
- Menos sensible a valores extremos que el MSE.

**Ejemplo en Python:**

```python
from sklearn.metrics import mean_absolute_error

# Cálculo del MAE
mae = mean_absolute_error(Y_real, Y_pred)
print(f"Error Absoluto Medio (MAE): {mae}")
```

**INDICADOR DE AJUSTE $R^2$ (COEFICIENTE DE DETERMINACIÓN).**

El $R^2$ mide qué tan bien el modelo explica la variabilidad de la variable dependiente.

Fórmula:

$$R^2 = 1 - \frac{\sum (Y_i - \hat{Y}_i)^2}{\sum (Y_i - \bar{Y})^2}$$

Interpretación

$R^2 = 1$: El modelo explica el 100% de la varianza.

$R^2$ = 0: El modelo no explica nada de la varianza.

$R^2$ < 0: El modelo es peor que una pediccion basada en la media de los datos

**Ejemplo en Python**

```python
from sklearn.metrics import r2_score

# Cálculo del R²
r2 = r2_score(Y_real, Y_pred)
print(f"Coeficiente de determinación R²: {r2}")
```

**IMPLEMENTACIÓN EN PYTHON CON LIBRERÍA STATSMODELS.**

En la práctica, se pueden usar bibliotecas como Statsmodels para construir y evaluar modelos de regresión lineal con métricas de error integradas.

**Ejemplo en Python:**

```python
import numpy as np
import statsmodels.api as sm

# Datos de ejemplo
X = np.array([1, 2, 3, 4, 5])
Y = np.array([2, 3, 5, 6, 8])

# Agregar una columna de 1s para el intercepto
X = sm.add_constant(X)

# Crear el modelo
modelo = sm.OLS(Y, X).fit()

# Resumen del modelo
print(modelo.summary())
```

Salida esperada:

- Valores de coeficientes ( $\boldsymbol{\beta_0}$ y $\boldsymbol{\beta_1}$ ).
- $R^2$ y $R^2$ ajustado.
- Errores estándar de los coeficientes.
- Pruebas de significancia estadística (p-values, t-values).

**REGRESIONES LINEALES MÚLTIPLES: QUÉ ES UNA REGRESIÓN LINEAL MÚLTIPLE.**

**¿Qué es una Regresión Lineal Múltiple?**

La Regresión Lineal Múltiple es una extensión de la regresión lineal simple que permite modelar la relación entre una variable dependiente (Y) y dos o más variables independientes (X1, X2,..., Xn)

Ecuación:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_n X_n + \epsilon$$

Componentes:

$\boldsymbol{\beta_0}$ dvf(Intercepto): Valor de Y cuando todas las variables independientes son 0.

$\boldsymbol{\beta_1}, \boldsymbol{\beta_2}, \ldots, \boldsymbol{\beta_n}$ (Coeficientes): Miden el efecto de cada variable independiente sobre Y.

**Ɛ** (Error residual): Captura variaciones no explicadas por el modelo.

**Ejemplo de Aplicación:**

Predecir las ventas (Y) de una empresa en función del gasto en publicidad en TV, radio e internet ($\boldsymbol{X_1}, \boldsymbol{X_2}, \boldsymbol{X_3}$ ).

## SUPUESTOS DE UNA REGRESIÓN MÚLTIPLE

Para que el modelo sea válido, se deben cumplir ciertos supuestos:

1. Linealidad: La relación entre las variables independientes y la variable dependiente debe ser lineal. Se puede verificar con gráficos de dispersión o transformaciones de variables.
2. Independencia de Errores: Los errores (ϵ) no deben estar correlacionados. Se evalúa mediante el test de Durbin-Watson.
3. Homocedasticidad: La varianza de los errores debe ser constante en todos los niveles de las variables independientes. Se verifica con gráficos de residuos.
4. Normalidad de Errores: Los errores deben seguir una distribución normal para que los intervalos de confianza y pruebas de hipótesis sean válidos. Se prueba con el test de Shapiro-Wilk o histogramas de residuos.

## MÉTODOS DE SELECCIÓN DEL MODELO

Dado que en una regresión múltiple pueden existir muchas variables, es importante seleccionar solo aquellas que aportan valor al modelo.

- Forward Selection (Selección Adelante): Se inicia con una variable y se añaden más en función de su significancia estadística.
- Backward Elimination (Eliminación Hacia Atrás): Se parte del modelo con todas las variables y se eliminan las menos significativas.
- Stepwise Regression (Regresión por Pasos): Combina Forward Selection y Backward Elimination para optimizar el modelo.

## INDICADOR R2-AJUSTADO

El coeficiente de determinación $R^2$ mide la proporción de la variabilidad de **Y** explicada por el modelo.

Sin embargo, en regresiones múltiples, agregar más variables siempre aumenta $R^2$, aunque estas no sean relevantes. Por ello, se utiliza el $R^2$ ajustado, que penaliza la inclusión de variables innecesarias.

Fórmula:

$$R^2_{\text{ajustado}} = 1 - \frac{(1-R^2)(n-1)}{n-k-1}$$

- Ŋ: Numero de observaciones.
- K: Numero de variables independientes.

**TEST DE NORMALIDAD DEL ERROR**

Para validar el supuesto de normalidad de los errores, se puede utilizar el test de Shapiro-Wilk.

**Ejemplo en Python:**

```python
from scipy.stats import shapiro
residuos = modelo.resid  # Residuos del modelo de regresión
stat, p_value = shapiro(residuos)

if p_value > 0.05:
    print("Los residuos siguen una distribución normal.")
else:
    print("Los residuos no siguen una distribución normal.")
```

**IMPLEMENTACIÓN EN PYTHON CON LIBRERÍA STATSMODELS**

La librería Statsmodels permite construir modelos de regresión múltiple y obtener métricas clave.

**Ejemplo en Python:**

```python
import numpy as np
import pandas as pd
import statsmodels.api as sm

# Datos de ejemplo: relación entre ventas y presupuesto en publicidad (TV, Radio, Internet)
data = pd.DataFrame({
    'Ventas': [200, 220, 250, 270, 300],
    'TV': [100, 120, 140, 160, 180],
    'Radio': [30, 40, 50, 60, 70],
    'Internet': [20, 25, 30, 35, 40]
})

# Variables dependiente e independientes
Y = data['Ventas']
X = data[['TV', 'Radio', 'Internet']]

# Agregar una columna de 1s para el intercepto
X = sm.add_constant(X)

# Crear el modelo de regresión
modelo = sm.OLS(Y, X).fit()

# Mostrar resumen del modelo
print(modelo.summary())
```

Salida esperada:

- Coeficientes $\boldsymbol{\beta_0}, \boldsymbol{\beta_1}, \boldsymbol{\beta_2}, \ldots$ .
- $R^2$ Ajustado para evaluar la caidad del modelo.
- Significancia de las variables mediante valores p.
- Pruebas de normalidad y homocedasticidad de los errores.

**EJERCICIO PRACTICO: REGRESIÓN LINEAL SIMPLE**

En este ejercicio práctico, aprenderás a ajustar una regresión lineal simple a un conjunto de datos y evaluar su rendimiento utilizando el coeficiente de determinación $R^2$ . Utilizaremos Python y la librería statsmodels para implementar el modelo.

**Paso 1: Importar Librerías**

Primero, importamos las librerías necesarias:

```
import pandas as pd
import statsmodels.api as sm
import matplotlib.pyplot as plt
```

**Paso 2: Crear un Conjunto de Datos**

Vamos a crear un conjunto de datos simple con una variable independiente (X) y una variable dependiente (Y):

```
# Crear un DataFrame con datos de ejemplo
df = pd.DataFrame({
    'X': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],  # Variable independiente
    'Y': [2, 4, 5, 4, 6, 8, 7, 9, 10, 12]  # Variable dependiente
})
```

**Paso 3: Visualizar los Datos**

Antes de ajustar el modelo, es útil visualizar la relación entre X e Y para tener una idea inicial de si existe una relación lineal.

```
# Gráfico de dispersión
plt.scatter(df['X'], df['Y'])
plt.title('Gráfico de Dispersión: X vs Y')
plt.xlabel('X (Variable Independiente)')
plt.ylabel('Y (Variable Dependiente)')
plt.grid(True)
plt.show()
```

Interpretación del Gráfico:

Si los puntos tienden a alinearse en una dirección, es probable que exista una relación lineal entre X e Y.

**Paso 4: Ajustar el Modelo de Regresión Lineal Simple**

Usamos la librería statsmodels para ajustar el modelo de regresión lineal simple.

```
# Añadir una constante para el intercepto (β0)
X = sm.add_constant(df['X'])  # X es la variable independiente
Y = df['Y']  # Y es la variable dependiente

# Ajustar el modelo de regresión lineal
modelo = sm.OLS(Y, X).fit()  # OLS: Ordinary Least Squares (Mínimos Cuadrados Ordinarios)
```

**Paso 5: Ver el Resumen del Modelo**

El resumen del modelo proporciona información detallada sobre el ajuste, incluyendo los coeficientes de regresión y el valor de $R^2$

```
print(modelo.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      Y   R-squared:                       0.945
Model:                            OLS   Adj. R-squared:                  0.938
Method:                 Least Squares   F-statistic:                     137.5
Date:                [Fecha y hora]     Prob (F-statistic):           1.23e-06
Time:                        [Hora]     Log-Likelihood:                -10.234
No. Observations:                  10   AIC:                             24.47
Df Residuals:                       8   BIC:                             25.07
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          1.2000      0.548      2.190      0.060      -0.068       2.468
X              1.0364      0.088     11.726      0.000       0.833       1.240
==============================================================================
Omnibus:                        0.123   Durbin-Watson:                   1.456
Prob(Omnibus):                  0.941   Jarque-Bera (JB):                0.260
Skew:                          -0.156   Prob(JB):                        0.878
Kurtosis:                       2.167   Cond. No.                         11.7
==============================================================================
```

**Paso 6: Interpretación del Modelo**

Coeficientes:

- Intercepto $(\boldsymbol{\beta_0})$**:** 1.200
  Es el valor predicho de Y cuando X = 0

- Pendiente $(\boldsymbol{\beta_1}\ )$**:** 1.0364
  Indica que por cada unidad que aumenta X, Y aumenta en 1.0364 unidades.

Coeficientes de Determinación ($R^2$):

- $R^2$ = 0.945
  - Significa que el 94.5% de la variabilidad en Y es explicada por X.
  - Un valor cercano a 1 indica un buen ajuste del modelo.

Valor ρ:

- Para la variable X, el valor ρ es 0.000, lo que indica que X es significativa para predecir Y.

**Paso 7: Visualizar la Línea de Regresión**

Para entender mejor el modelo, graficamos la línea de regresión sobre los datos originales.

```
# Graficar los datos originales
plt.scatter(df['X'], df['Y'], label='Datos Observados')

# Graficar la línea de regresión
plt.plot(df['X'], modelo.predict(X), color='red', label='Línea de Regresión')

# Añadir etiquetas y leyenda
plt.title('Regresión Lineal Simple: X vs Y')
plt.xlabel('X (Variable Independiente)')
plt.ylabel('Y (Variable Dependiente)')
plt.legend()
plt.grid(True)
plt.show()
```

Interpretación del Gráfico:

- La línea roja representa la relación lineal estimada entre X y Y.
- Si los puntos están cerca de la línea, el modelo tiene un buen ajuste.

**Paso 8: Evaluar el Rendimiento del Modelo**

Además de $R^2$, podemos calcular otras métricas de error, como el Error Cuadrático Medio (MSE) y el Error Absoluto Medio (MAE).

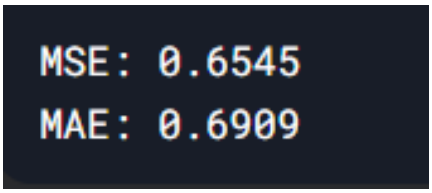
```python
from sklearn.metrics import mean_squared_error, mean_absolute_error

# Calcular predicciones
Y_pred = modelo.predict(X)

# Calcular MSE y MAE
mse = mean_squared_error(Y, Y_pred)
mae = mean_absolute_error(Y, Y_pred)

print(f"MSE: {mse}")
print(f"MAE: {mae}")
```

Salida




**Conclusión del Ejercicio**

- Ajustamos un modelo de regresión lineal simple a un conjunto de datos y evaluamos su rendimiento utilizando $R^2$ , MSE y MAE.
- El modelo tiene un buen ajuste ($R^2$= 0.945), lo que indica que la variable X explica la mayor parte de la variabilidad en Y.
- Este ejercicio es un punto de partida para entender cómo funcionan las regresiones lineales y cómo evaluar su rendimiento.