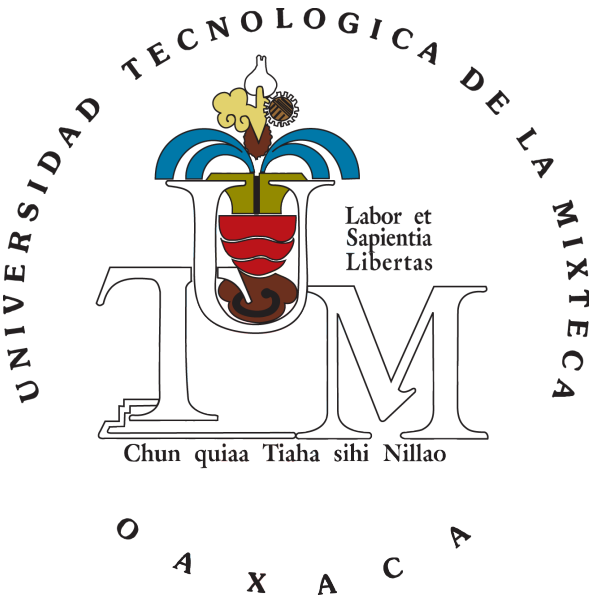


Estadistica con Python

Regresión Lineal Simple

Ana Delia Olvera Cervantes

Maestría en Ciencia de Datos.
Melchor Nolasco Cosijoeza
Grupo: Propedeutico

17 de septiembre de 2024

RegresionLinealSimple

September 16, 2024

Cosijoeza Melchor Nolasco

1 REGRESION LINEAL SIMPLE

Suponga que un analista de deportes quiere saber si exste una relacion entre el numero de veces que batean los jugadores de un equipo de beisbol y el numero de runs que consigue. En caso de existir y de establecer el modelo, podría predecir el resultado del partido.

1.1 Librerias para tratamiento de datos

[ ]:
```
import pandas as pd
import numpy as np
```

1.2 Librerias para los graficos

[ ]:
```
import matplotlib.pyplot as plt
from matplotlib import style
import seaborn as sns
```

1.3 Librerias para el procesado y el modelado

[ ]:
```
from scipy.stats import pearsonr
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
import statsmodels.api as sm
import statsmodels.formula.api as smf
```

1.4 Configuración de Matplotlib

[ ]:
```
plt.rcParams['image.cmap'] = 'bwr'
plt.rcParams['savefig.bbox'] = 'tight'
style.use('ggplot') or plt.style.use('ggplot')
```

1.5 Configuaración de Warnings

[ ]:
```
import warnings
warnings.filterwarnings('ignore')
```

1.6 Datos

[ ]:
```
equipos =[
    "Texas","Boston","Detroit","Kansas","St.","New_S.","New_Y.
 ↪","Milwaukee","Colorado",
    "Houston","Baltimore","Los_An.","Chicago","Cincinnati","Los_P.
 ↪","Philadelphia",
   ␣
 ↪"Chicago","Cleveland","Arizona","Toronto","Minnesota","Florida","Pittsburgh",
    "Oakland","Tampa","Atlanta","Washington","San.F","San.I","Seattle"
]
bateos = [
    5659,5710,5563,5672,5532,5600,5518,5447,5544,5598,
    5585,5436,5549,5612,5513,5579,5502,5509,5421,5559,
    5487,5508,5421,5452,5436,5528,5441,5486,5417,5421]
runs = [
    855,875,787,730,762,718,867,721,735,615,708,644,654,735,
    667,713,654,704,731,743,619,625,610,645,707,641,624,570,593,556
]

datos = pd.DataFrame({
    "equipos":equipos,
    "bateos":bateos,
    "runs":runs
})
datos.head()
```

[ ]:

	equipos	bateos	runs
0	Texas	5659	855
1	Boston	5710	875
2	Detroit	5563	787
3	Kansas	5672	730
4	St.	5532	762

2 Representación Gráfica

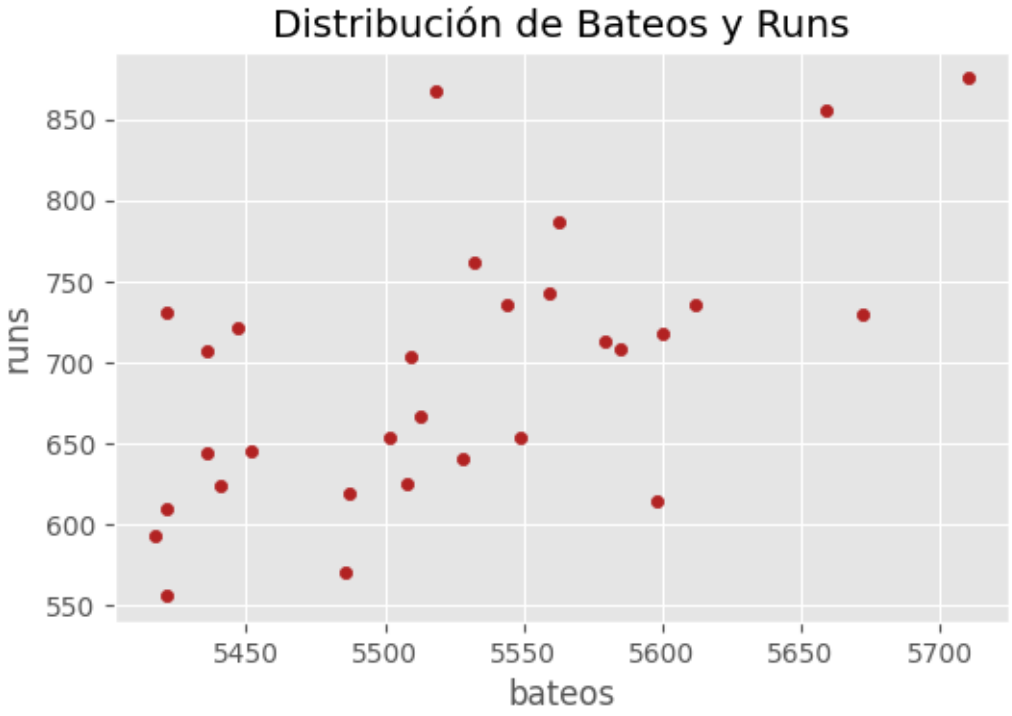
Representar los datos para poder intuir si existe una realacion y cuantificar la relacion mediante un coeficiente de correlación.

[ ]:
```
fig, ax = plt.subplots(figsize=(6,3.84))
datos.plot(
    x = "bateos",
```

```
    y = "runs",
    c = "firebrick",
    kind = "scatter",
    ax = ax,
)

ax.set_title("Distribución de Bateos y Runs")
```

[ ]: Text(0.5, 1.0, 'Distribución de Bateos y Runs')



2.1 Correlacion lineal entre las dos variables

[ ]:
```
corr_test = pearsonr(x = datos["bateos"],y=datos["runs"])
print("Coeficiente de correlación de Pearson: ",corr_test[0])
print("P-value: ",corr_test[1])
print(corr_test)
```

```
Coeficiente de correlación de Pearson:  0.6106270467206688
P-value:  0.0003388351359791978
PearsonRResult(statistic=0.6106270467206688, pvalue=0.0003388351359791978)
```

3 Ajuste del modelo

Se ajusta el modelo empleando como variable respuesta **runs** y como predictor **bateos**. Como en todo estudio predictivo, no solo es importante ajustar el modelo , sino tambien cuantificar su capacidad para predecir nuevas observaciones. Para poder hacer esta evaluacion, se dividen los datos en dos grupos, uno de entrenamiento y otro de test.

3.1 División de los datos en train y test.

[ ]:
```
x = datos[["bateos"]]
y = datos["runs"]
x_train,x_test,y_train,y_test = train_test_split(
    x.values.reshape(-1,1),
    y.values.reshape(-1,1),
    train_size = 0.8,
    random_state = 1234,
    shuffle = True
)
```

3.1.1 Creación del modelo

[ ]:
```
modelo = LinearRegression()
modelo.fit(X = x_train.reshape(-1,1),y = y_train)
```

[ ]:
```
LinearRegression()
```

3.2 Informacion del Modelo

[ ]:
```
print("Intercept:",modelo.intercept_)
print("Coeficiente:",list(zip(x.columns,modelo.coef_.flatten(),)))
print("Coeficiente de determinación R^2",modelo.score(x,y))
```

```
Intercept: [-2367.7028413]
Coeficiente: [('bateos', 0.5528713534479736)]
Coeficiente de determinación R^2 0.3586119899498744
```

Evaluar la capacidad predictiva empleando el conjunto test.

3.3 Error de test del Modelo

Evaluamos la capacidad predictiva empleando el conjunto de test.

[ ]:
```
predicciones = modelo.predict(X = x_test)
print(predicciones[0:3,])
rmse = mean_squared_error(
    y_true = y_test,
    y_pred = predicciones,
    squared = False
```

```
    )
print(f"El error (rmse) de test es: {rmse}")
```

```
[[643.78742093]
 [720.0836677 ]
 [690.78148597]]
El error (rmse) de test es: 59.336716083360486
```

4 Implementación con Statsmodels

4.1 Division de los datos en train y test

[ ]:
```
x = datos[["bateos"]]
y = datos["runs"]

x_train,x_test,y_train,y_test = train_test_split(
    x.values.reshape(-1,1),
    y.values.reshape(-1,1),
    train_size = 0.8,
    random_state = 1234,
    shuffle = True
)
```

4.2 Creacion del modelo utilizando el modo formula (similar a R)

[ ]:
```
datos_train = pd.DataFrame(np.hstack((x_train,y_train)),columns =␣
 ↪["bateos","runs"])
modelo = smf.ols(formula = "runs ~ bateos",data = datos_train)
modelo = modelo.fit()
print(modelo.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                   runs   R-squared:                       0.271
Model:                            OLS   Adj. R-squared:                  0.238
Method:                 Least Squares   F-statistic:                     8.191
Date:                Mon, 16 Sep 2024   Prob (F-statistic):            0.00906
Time:                        08:21:16   Log-Likelihood:                -134.71
No. Observations:                  24   AIC:                             273.4
Df Residuals:                      22   BIC:                             275.8
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept  -2367.7028   1066.357     -2.220      0.037   -4579.192    -156.214
bateos         0.5529      0.193      2.862      0.009       0.152       0.953
```

```
==============================================================================
Omnibus:                        5.033   Durbin-Watson:                   1.902
Prob(Omnibus):                  0.081   Jarque-Bera (JB):                3.170
Skew:                           0.829   Prob(JB):                        0.205
Kurtosis:                       3.650   Cond. No.                     4.17e+05
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 4.17e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
```

4.3 Creacion del modelo utilizando matrices como en scikitLearn

A la matriz de predictores se le tiene que añadir una columna de 1s para el intercept del modelo

[ ]:
```
x_train = sm.add_constant(x_train,prepend=True)
modelo = sm.OLS(endog = y_train,exog = x_train)
modelo = modelo.fit()
print(modelo.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.271
Model:                            OLS   Adj. R-squared:                  0.238
Method:                 Least Squares   F-statistic:                     8.191
Date:                Mon, 16 Sep 2024   Prob (F-statistic):            0.00906
Time:                        08:21:16   Log-Likelihood:                -134.71
No. Observations:                  24   AIC:                             273.4
Df Residuals:                      22   BIC:                             275.8
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const      -2367.7028   1066.357     -2.220      0.037   -4579.192    -156.214
x1             0.5529      0.193      2.862      0.009       0.152       0.953
==============================================================================
Omnibus:                        5.033   Durbin-Watson:                   1.902
Prob(Omnibus):                  0.081   Jarque-Bera (JB):                3.170
Skew:                           0.829   Prob(JB):                        0.205
Kurtosis:                       3.650   Cond. No.                     4.17e+05
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

```
[2] The condition number is large, 4.17e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
```

4.4 Intervalos de confianza de los coeficientes

Intervalos de confianza para los coeficientes del modelo

[ ]:
```
modelo.conf_int(alpha=0.05)
```

[ ]:
```
array([[-4.57919205e+03, -1.56213633e+02],
       [ 1.52244180e-01,  9.53498527e-01]])
```

4.5 Predicciones con intervalo de confianza del 95%

[ ]:
```
predicciones = modelo.get_prediction(exog = x_train).summary_frame(alpha=0.05)
predicciones.head(5)
```

[ ]:

	mean	mean_se	mean_ci_lower	mean_ci_upper	obs_ci_lower	obs_ci_upper
0	768.183475	32.658268	700.454374	835.912577	609.456054	926.910897
1	646.551778	19.237651	606.655332	686.448224	497.558860	795.544695
2	680.276930	14.186441	650.856053	709.697807	533.741095	826.812765
3	735.011194	22.767596	687.794091	782.228298	583.893300	886.129088
4	629.412766	23.713237	580.234522	678.591009	477.670673	781.154858

5 Representación grafica del modelo

5.1 Predicciones con intervalo de confianza del 95%

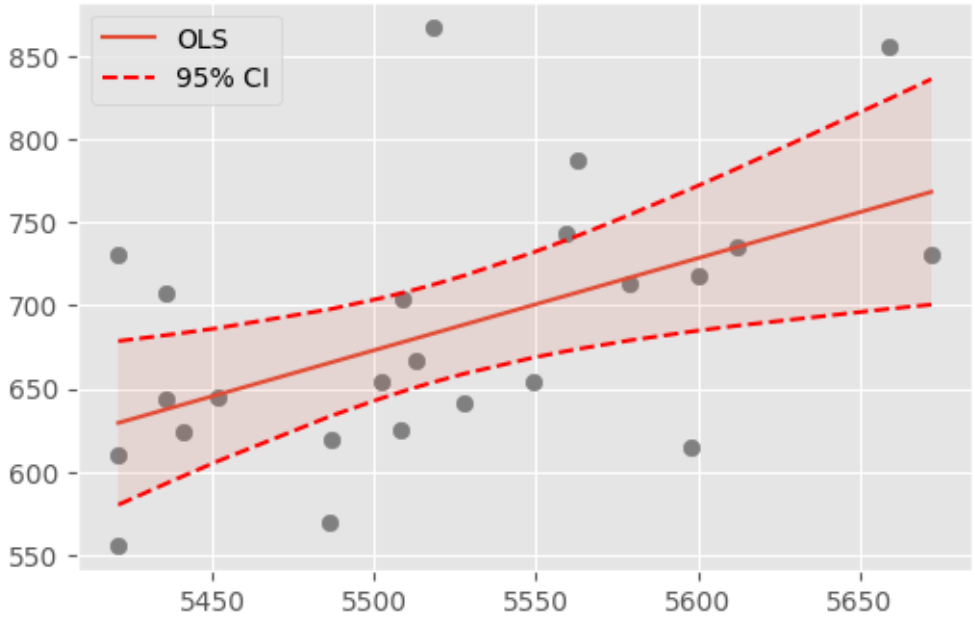
[ ]:
```
predicciones = modelo.get_prediction(exog = x_train).summary_frame(alpha=0.05)
predicciones["x"] = x_train[:,1]
predicciones["y"] = y_train
predicciones = predicciones.sort_values("x")
```

5.2 Grafico del modelo

[ ]:
```
fig,ax = plt.subplots(figsize=(6,3.84))
ax.scatter(predicciones["x"],predicciones["y"],marker="o",color="gray")
ax.plot(predicciones["x"],predicciones["mean"],linestyle="-",label="OLS")
```

```
ax.
 ↪plot(predicciones["x"],predicciones["mean_ci_lower"],linestyle="--",color="red",label="95%␣
 ↪CI")
ax.
 ↪plot(predicciones["x"],predicciones["mean_ci_upper"],linestyle="--",color="red")
ax.fill_between(
    x = predicciones["x"],
    y1 = predicciones["mean_ci_lower"],
    y2 = predicciones["mean_ci_upper"],
    alpha = 0.1
)
ax.legend()
```

[ ]: <matplotlib.legend.Legend at 0x7887828e5540>



6 Error de test del modelo

[ ]:
```
x_test = sm.add_constant(x_test,prepend=True)
predicciones = modelo.predict(exog = x_test)
rmse = mean_squared_error(
    y_true = y_test,
    y_pred = predicciones,
    squared = False
    )
print(f"El error (rmse) de test es: {rmse}")
```

```
El error (rmse) de test es: 59.33671608336119
```