pr2-temp-gs

November 7, 2024

[ ]:
```
pip install pandas
```

```
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages
(2.2.2)
Requirement already satisfied: numpy>=1.22.4 in /usr/local/lib/python3.10/dist-
packages (from pandas) (1.26.4)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.10/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-
packages (from pandas) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-
packages (from pandas) (2024.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-
packages (from python-dateutil>=2.8.2->pandas) (1.16.0)
```

[ ]:
```
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import matplotlib.pyplot as plt
import seaborn as sns
```

[ ]:
```
df = pd.read_csv("temperatures.csv")
```

[ ]:
```
df
```

[ ]:
```
     YEAR    JAN    FEB    MAR    APR    MAY    JUN    JUL    AUG    SEP  \
0    1901  22.40  24.14  29.07  31.91  33.41  33.18  31.21  30.39  30.47
1    1902  24.93  26.58  29.77  31.78  33.73  32.91  30.92  30.73  29.80
2    1903  23.44  25.03  27.83  31.39  32.91  33.00  31.34  29.98  29.85
3    1904  22.50  24.73  28.21  32.02  32.64  32.07  30.36  30.09  30.04
4    1905  22.00  22.83  26.68  30.01  33.32  33.25  31.44  30.68  30.12
..    …      …      …      …      …      …      …      …      …      …
112  2013  24.56  26.59  30.62  32.66  34.46  32.44  31.07  30.76  31.04
113  2014  23.83  25.97  28.95  32.74  33.77  34.15  31.85  31.32  30.68
114  2015  24.58  26.89  29.07  31.87  34.09  32.48  31.88  31.52  31.55
115  2016  26.94  29.72  32.62  35.38  35.72  34.03  31.64  31.79  31.66
116  2017  26.45  29.46  31.60  34.95  35.84  33.82  31.88  31.72  32.22
```

```
       OCT    NOV    DEC  ANNUAL  JAN-FEB  MAR-MAY  JUN-SEP  OCT-DEC
0    29.97  27.31  24.49   28.96    23.27    31.46    31.27    27.25
1    29.12  26.31  24.04   29.22    25.75    31.76    31.09    26.49
2    29.04  26.08  23.65   28.47    24.24    30.71    30.92    26.26
3    29.20  26.36  23.63   28.49    23.62    30.95    30.66    26.40
4    30.67  27.52  23.82   28.30    22.25    30.00    31.33    26.57
..     …      …     …       …        …        …        …
112  30.27  27.83  25.37   29.81    25.58    32.58    31.33    27.83
113  30.29  28.05  25.08   29.72    24.90    31.82    32.00    27.81
114  31.04  28.10  25.67   29.90    25.74    31.68    31.87    28.27
115  31.98  30.11  28.01   31.63    28.33    34.57    32.28    30.03
116  32.29  29.60  27.18   31.42    27.95    34.13    32.41    29.69

[117 rows x 18 columns]
```

This function provides summary statistics for each numerical column by default, though it can also be used for categorical data if specified.

df.describe()

[ ]:
```
df.describe()
```

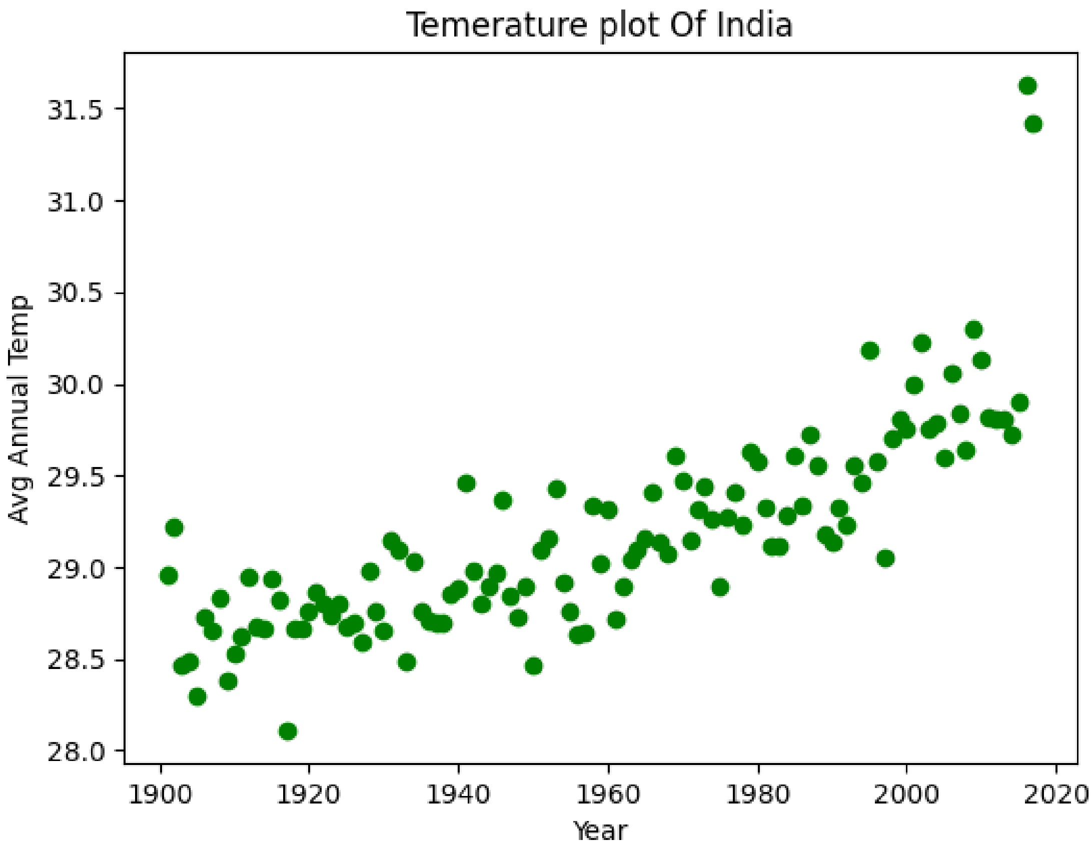
[ ]:
```
              YEAR         JAN         FEB         MAR         APR  \
count   117.000000  117.000000  117.000000  117.000000  117.000000
mean   1959.000000   23.687436   25.597863   29.085983   31.975812
std      33.919021    0.834588    1.150757    1.068451    0.889478
min    1901.000000   22.000000   22.830000   26.680000   30.010000
25%    1930.000000   23.100000   24.780000   28.370000   31.460000
50%    1959.000000   23.680000   25.480000   29.040000   31.950000
75%    1988.000000   24.180000   26.310000   29.610000   32.420000
max    2017.000000   26.940000   29.720000   32.620000   35.380000

              MAY         JUN         JUL         AUG         SEP         OCT  \
count  117.000000  117.000000  117.000000  117.000000  117.000000  117.000000
mean    33.565299   32.774274   31.035897   30.507692   30.486752   29.766581
std      0.724905    0.633132    0.468818    0.476312    0.544295    0.705492
min     31.930000   31.100000   29.760000   29.310000   29.070000   27.900000
25%     33.110000   32.340000   30.740000   30.180000   30.120000   29.380000
50%     33.510000   32.730000   31.000000   30.540000   30.520000   29.780000
75%     34.030000   33.180000   31.330000   30.760000   30.810000   30.170000
max     35.840000   34.480000   32.760000   31.840000   32.220000   32.290000

              NOV         DEC      ANNUAL     JAN-FEB     MAR-MAY     JUN-SEP  \
count  117.000000  117.000000  117.000000  117.000000  117.000000  117.000000
mean    27.285470   24.608291   29.181368   24.629573   31.517607   31.198205
std      0.714518    0.782644    0.555555    0.911239    0.740585    0.420508
min     25.700000   23.020000   28.110000   22.250000   29.920000   30.240000
```

```
25%      26.790000   24.040000   28.760000   24.110000   31.040000   30.920000
50%      27.300000   24.660000   29.090000   24.530000   31.470000   31.190000
75%      27.720000   25.110000   29.470000   25.150000   31.890000   31.400000
max      30.110000   28.010000   31.630000   28.330000   34.570000   32.410000

            OCT-DEC
count    117.000000
mean      27.208120
std        0.672003
min       25.740000
25%       26.700000
50%       27.210000
75%       27.610000
max       30.030000
```

[ ]:
```
#input data
x = df['YEAR']

#output data
y = df['ANNUAL']
```

[ ]:
```
plt.title("Temerature plot Of India")
plt.xlabel("Year")
plt.ylabel("Avg Annual Temp")
plt.scatter(x,y, color="green")
```

[ ]:
```
<matplotlib.collections.PathCollection at 0x7bf62aa88ac0>
```



[ ]:
```
x.shape
```

[ ]:
```
(117,)
```

[ ]:
```
x=x.values
```

So after running x = x.values, x will now be a one-dimensional NumPy array rather than a pandas Series, making it easier to work with certain machine learning models in Scikit-learn that expect inputs as arrays.

[ ]:
```
x
```

[ ]:
```
array([1901, 1902, 1903, 1904, 1905, 1906, 1907, 1908, 1909, 1910, 1911,
       1912, 1913, 1914, 1915, 1916, 1917, 1918, 1919, 1920, 1921, 1922,
       1923, 1924, 1925, 1926, 1927, 1928, 1929, 1930, 1931, 1932, 1933,
       1934, 1935, 1936, 1937, 1938, 1939, 1940, 1941, 1942, 1943, 1944,
       1945, 1946, 1947, 1948, 1949, 1950, 1951, 1952, 1953, 1954, 1955,
       1956, 1957, 1958, 1959, 1960, 1961, 1962, 1963, 1964, 1965, 1966,
       1967, 1968, 1969, 1970, 1971, 1972, 1973, 1974, 1975, 1976, 1977,
       1978, 1979, 1980, 1981, 1982, 1983, 1984, 1985, 1986, 1987, 1988,
```

```
       1989, 1990, 1991, 1992, 1993, 1994, 1995, 1996, 1997, 1998, 1999,
       2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010,
       2011, 2012, 2013, 2014, 2015, 2016, 2017])
```

[ ]:
```
x=x.reshape(117,1)
```

[ ]:
```
x.shape
```

[ ]:
```
(117, 1)
```

[ ]:
```
x
```

[ ]:
```
array([[1901],
       [1902],
       [1903],
       [1904],
       [1905],
       [1906],
       [1907],
       [1908],
       [1909],
       [1910],
       [1911],
       [1912],
       [1913],
       [1914],
       [1915],
       [1916],
       [1917],
       [1918],
       [1919],
       [1920],
       [1921],
       [1922],
       [1923],
       [1924],
       [1925],
       [1926],
       [1927],
       [1928],
       [1929],
       [1930],
       [1931],
       [1932],
       [1933],
       [1934],
       [1935],
```

[1936],
[1937],
[1938],
[1939],
[1940],
[1941],
[1942],
[1943],
[1944],
[1945],
[1946],
[1947],
[1948],
[1949],
[1950],
[1951],
[1952],
[1953],
[1954],
[1955],
[1956],
[1957],
[1958],
[1959],
[1960],
[1961],
[1962],
[1963],
[1964],
[1965],
[1966],
[1967],
[1968],
[1969],
[1970],
[1971],
[1972],
[1973],
[1974],
[1975],
[1976],
[1977],
[1978],
[1979],
[1980],
[1981],
[1982],

```
       [1983],
       [1984],
       [1985],
       [1986],
       [1987],
       [1988],
       [1989],
       [1990],
       [1991],
       [1992],
       [1993],
       [1994],
       [1995],
       [1996],
       [1997],
       [1998],
       [1999],
       [2000],
       [2001],
       [2002],
       [2003],
       [2004],
       [2005],
       [2006],
       [2007],
       [2008],
       [2009],
       [2010],
       [2011],
       [2012],
       [2013],
       [2014],
       [2015],
       [2016],
       [2017]])
```

[ ]:
```
y.shape
```

[ ]: (117,)

[ ]:
```
y=y.values
```

[ ]:
```
y=y.reshape(117,)
```

[ ]:
```
y
```

[ ]:
```
array([28.96, 29.22, 28.47, 28.49, 28.3 , 28.73, 28.65, 28.83, 28.38,
       28.53, 28.62, 28.95, 28.67, 28.66, 28.94, 28.82, 28.11, 28.66,
       28.66, 28.76, 28.86, 28.8 , 28.74, 28.8 , 28.67, 28.7 , 28.59,
       28.98, 28.76, 28.65, 29.15, 29.09, 28.49, 29.03, 28.76, 28.71,
       28.7 , 28.7 , 28.85, 28.88, 29.46, 28.98, 28.8 , 28.89, 28.97,
       29.37, 28.84, 28.73, 28.89, 28.47, 29.09, 29.16, 29.43, 28.92,
       28.76, 28.63, 28.64, 29.34, 29.02, 29.31, 28.72, 28.89, 29.04,
       29.09, 29.16, 29.41, 29.14, 29.07, 29.61, 29.47, 29.15, 29.31,
       29.44, 29.26, 28.89, 29.27, 29.41, 29.23, 29.63, 29.58, 29.32,
       29.12, 29.11, 29.28, 29.61, 29.33, 29.72, 29.55, 29.18, 29.14,
       29.32, 29.23, 29.55, 29.46, 30.18, 29.58, 29.05, 29.7 , 29.81,
       29.75, 29.99, 30.23, 29.75, 29.79, 29.6 , 30.06, 29.84, 29.64,
       30.3 , 30.13, 29.82, 29.81, 29.81, 29.72, 29.9 , 31.63, 31.42])
```

[ ]:
```
regressor = LinearRegression()
```

[ ]:
```
regressor.fit(x,y)
```

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

y = m.x + c

m = slope

c = y -intercep

where m is the gradient of the line (how steep the line is) and c is the y -intercept (the point in which the line crosses the y -axis)

regressor.coef_ tells you how much the target variable (y) changes with a one-unit change in the feature(s) (x).

regressor.intercept_ is the value of y when x is zero. It's where the line "intersects" (or crosses) the y-axis on the graph. In short, the intercept is the starting point of your prediction when there's no input value (x = 0).

[ ]:
```
regressor.coef_ #m
```

[ ]:
```
array([[0.01312158]])
```

[ ]:
```
regressor.intercept_ #c
```

[ ]:
```
3.4761897126187016
```

[ ]:
```
regressor.predict([[2035]])
```

[ ]:
```
array([[30.1786077]])
```

regressor.predict(…): This applies the learned linear regression model (which you trained with regressor.fit(x, y)) to the input data 2035. It calculates the predicted value of the target variable

(y) based on the linear equation determined during training (using the model's coefficients and intercept).

```
[ ]: predicted = regressor.predict(x)
```

.predict(x): This method takes the input features (x) and uses the learned model to make predictions. It applies the equation of the line that was determined during training (which involves the regressor.coef_ and regressor.intercept_) to calculate the predicted values for y.

predicted: This is the output of the .predict(x) method. It contains the model's predicted values of y for the given x values.

```
[ ]: predicted
```

```
[ ]: array([28.4203158 , 28.43343739, 28.44655897, 28.45968055, 28.47280213,
            28.48592371, 28.49904529, 28.51216687, 28.52528846, 28.53841004,
            28.55153162, 28.5646532 , 28.57777478, 28.59089636, 28.60401794,
            28.61713952, 28.63026111, 28.64338269, 28.65650427, 28.66962585,
            28.68274743, 28.69586901, 28.70899059, 28.72211218, 28.73523376,
            28.74835534, 28.76147692, 28.7745985 , 28.78772008, 28.80084166,
            28.81396324, 28.82708483, 28.84020641, 28.85332799, 28.86644957,
            28.87957115, 28.89269273, 28.90581431, 28.91893589, 28.93205748,
            28.94517906, 28.95830064, 28.97142222, 28.9845438 , 28.99766538,
            29.01078696, 29.02390855, 29.03703013, 29.05015171, 29.06327329,
            29.07639487, 29.08951645, 29.10263803, 29.11575961, 29.1288812 ,
            29.14200278, 29.15512436, 29.16824594, 29.18136752, 29.1944891 ,
            29.20761068, 29.22073227, 29.23385385, 29.24697543, 29.26009701,
            29.27321859, 29.28634017, 29.29946175, 29.31258333, 29.32570492,
            29.3388265 , 29.35194808, 29.36506966, 29.37819124, 29.39131282,
            29.4044344 , 29.41755599, 29.43067757, 29.44379915, 29.45692073,
            29.47004231, 29.48316389, 29.49628547, 29.50940705, 29.52252864,
            29.53565022, 29.5487718 , 29.56189338, 29.57501496, 29.58813654,
            29.60125812, 29.6143797 , 29.62750129, 29.64062287, 29.65374445,
            29.66686603, 29.67998761, 29.69310919, 29.70623077, 29.71935236,
            29.73247394, 29.74559552, 29.7587171 , 29.77183868, 29.78496026,
            29.79808184, 29.81120342, 29.82432501, 29.83744659, 29.85056817,
            29.86368975, 29.87681133, 29.88993291, 29.90305449, 29.91617608,
            29.92929766, 29.94241924])
```

```
[ ]: y
```

```
[ ]: array([28.96, 29.22, 28.47, 28.49, 28.3 , 28.73, 28.65, 28.83, 28.38,
            28.53, 28.62, 28.95, 28.67, 28.66, 28.94, 28.82, 28.11, 28.66,
            28.66, 28.76, 28.86, 28.8 , 28.74, 28.8 , 28.67, 28.7 , 28.59,
            28.98, 28.76, 28.65, 29.15, 29.09, 28.49, 29.03, 28.76, 28.71,
            28.7 , 28.7 , 28.85, 28.88, 29.46, 28.98, 28.8 , 28.89, 28.97,
            29.37, 28.84, 28.73, 28.89, 28.47, 29.09, 29.16, 29.43, 28.92,
            28.76, 28.63, 28.64, 29.34, 29.02, 29.31, 28.72, 28.89, 29.04,
```

```
         29.09, 29.16, 29.41, 29.14, 29.07, 29.61, 29.47, 29.15, 29.31,
         29.44, 29.26, 28.89, 29.27, 29.41, 29.23, 29.63, 29.58, 29.32,
         29.12, 29.11, 29.28, 29.61, 29.33, 29.72, 29.55, 29.18, 29.14,
         29.32, 29.23, 29.55, 29.46, 30.18, 29.58, 29.05, 29.7 , 29.81,
         29.75, 29.99, 30.23, 29.75, 29.79, 29.6 , 30.06, 29.84, 29.64,
         30.3 , 30.13, 29.82, 29.81, 29.81, 29.72, 29.9 , 31.63, 31.42])
```

[ ]:
```
#Mean absolute Error
abs(y-predicted)
```

[ ]:
```
array([0.5396842 , 0.78656261, 0.02344103, 0.03031945, 0.17280213,
       0.24407629, 0.15095471, 0.31783313, 0.14528846, 0.00841004,
       0.06846838, 0.3853468 , 0.09222522, 0.06910364, 0.33598206,
       0.20286048, 0.52026111, 0.01661731, 0.00349573, 0.09037415,
       0.17725257, 0.10413099, 0.03100941, 0.07788782, 0.06523376,
       0.04835534, 0.17147692, 0.2054015 , 0.02772008, 0.15084166,
       0.33603676, 0.26291517, 0.35020641, 0.17667201, 0.10644957,
       0.16957115, 0.19269273, 0.20581431, 0.06893589, 0.05205748,
       0.51482094, 0.02169936, 0.17142222, 0.0945438 , 0.02766538,
       0.35921304, 0.18390855, 0.30703013, 0.16015171, 0.59327329,
       0.01360513, 0.07048355, 0.32736197, 0.19575961, 0.3688812 ,
       0.51200278, 0.51512436, 0.17175406, 0.16136752, 0.1155109 ,
       0.48761068, 0.33073227, 0.19385385, 0.15697543, 0.10009701,
       0.13678141, 0.14634017, 0.22946175, 0.29741667, 0.14429508,
       0.1888265 , 0.04194808, 0.07493034, 0.11819124, 0.50131282,
       0.1344344 , 0.00755599, 0.20067757, 0.18620085, 0.12307927,
       0.15004231, 0.36316389, 0.38628547, 0.22940705, 0.08747136,
       0.20565022, 0.1712282 , 0.01189338, 0.39501496, 0.44813654,
       0.28125812, 0.3843797 , 0.07750129, 0.18062287, 0.52625555,
       0.08686603, 0.62998761, 0.00689081, 0.10376923, 0.03064764,
       0.25752606, 0.48440448, 0.0087171 , 0.01816132, 0.18496026,
       0.26191816, 0.02879658, 0.18432501, 0.46255341, 0.27943183,
       0.04368975, 0.06681133, 0.07993291, 0.18305449, 0.01617608,
       1.70070234, 1.47758076])
```

[ ]:
```
np.mean(abs(y-predicted))
```

[ ]:
```
0.22535284978630413
```

[ ]:
```
mean_absolute_error(y,predicted)
```

[ ]:
```
0.22535284978630413
```

[ ]:
```
np.mean((y-predicted)**2)
```

[ ]:
```
0.10960795229110352
```

[ ]:
```
mean_squared_error(y,predicted)
```

[ ]: 0.10960795229110352

[ ]:
```
r2_score(y,predicted)
```

[ ]: 0.6418078912783682

[ ]:
```
regressor.score(x,y)
```

[ ]: 0.6418078912783682

The f before the string tells Python to evaluate the expression inside the curly braces {}. The mean_squared_error(y, predicted) function is called, and its result is placed inside the string where {} is.

MAE = (1/n) * Σ|y_true(i) - y_pred(i)|

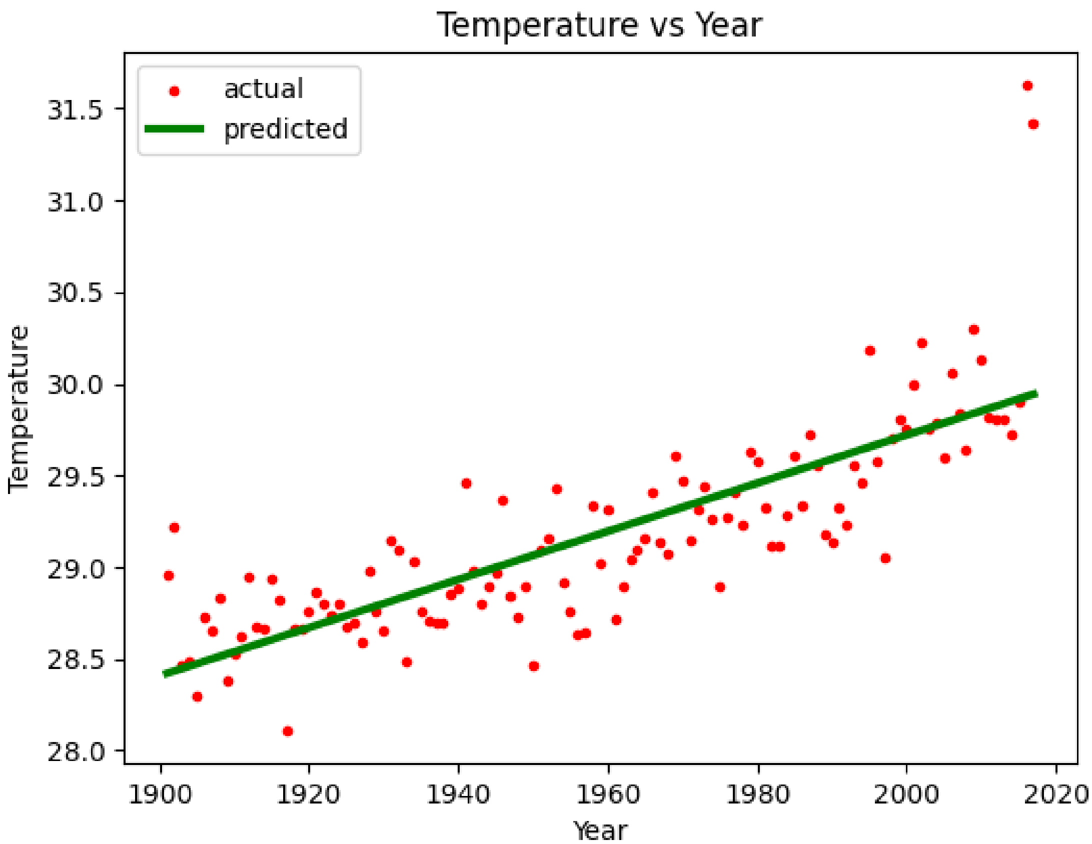
MSE = (1/n) * Σ(y_true(i) - y_pred(i))^2

R^2: R^2 = 1 - (Σ(y_true(i) - y_pred(i))^2 / Σ(y_true(i) - mean(y_true))^2)

[ ]:
```
print(f"MSE:  {mean_squared_error(y,predicted)}")
print(f"MAE:  {mean_absolute_error(y,predicted)}")
print(f"R-Sqaure :  {r2_score(y,predicted)}")
```

```
MSE:  0.10960795229110352
MAE:  0.22535284978630413
R-Sqaure :  0.6418078912783682
```

[ ]:
```
plt.scatter(x, y, label='actual',color='red',marker='.')
plt.plot(x, predicted,label='predicted', color='green', linewidth=3)
plt.title("Temperature vs Year")
plt.xlabel("Year")
plt.ylabel("Temperature")
plt.legend()
plt.show()
```



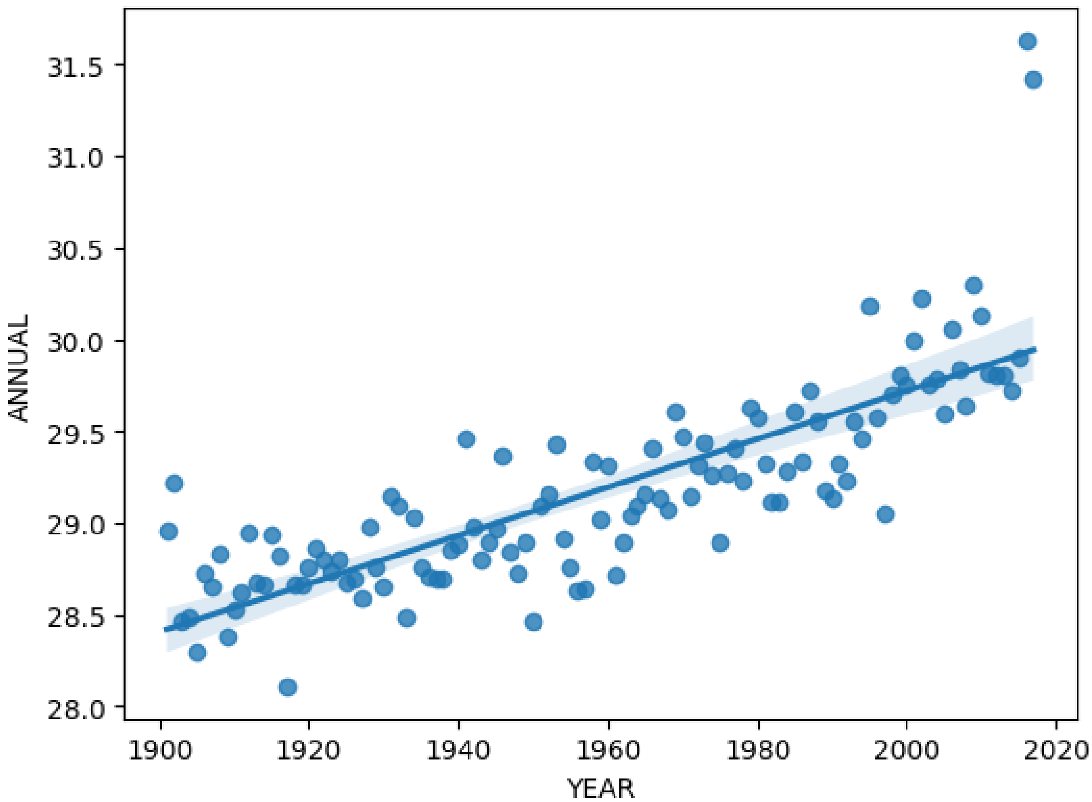
The line sns.regplot(x='YEAR', y='ANNUAL', data=df) is using Seaborn to create a scatter plot along with a linear regression line. Here's a detailed breakdown:

sns.regplot():

This function from Seaborn creates a scatter plot and fits a regression line (linear regression) to the data. It combines a scatter plot with a simple linear regression model, making it useful for visualizing the relationship between two variables.

[ ]:
```
sns.regplot(x='YEAR',y='ANNUAL',data=df)
```

[ ]:
```
<Axes: xlabel='YEAR', ylabel='ANNUAL'>
```



====================After x shape and y shape======================

x_train, x_test, y_train, y_test = train_test_split(x, y,test_size=0.25)

print(f“x Training dataset: {x_train.shape}”)

print(f“y Training dataset: {y_train.shape}”)

print(f“x test dataset: {x_test.shape}”)

print(f“y test dataset: {y_test.shape}”)

==

model.fit(x_train,y_train)

==

y_pred = model.predict(x_test)

===

plt.scatter(x_train, y_train, color=‘blue’)

plt.plot(x_test, y_pred, color=‘red’, linewidth=3)

plt.title(“Temperature vs Year”)

plt.xlabel(“Year”)

plt.ylabel(“Temperature”)

plt.show()

sns.regplot(data=df,x=x_train,y=y_train,)

```
[ ]:
```

```
[ ]:
```