Linear_Regression(1)

November 17, 2023

[1]:
```
import pandas as pd
# importing the pandas library as pd for data manipulation
```

[2]:
```
%matplotlib inline
```

[3]:
```
# importing the matplotlib library as plt for data visualization
import matplotlib.pyplot as plt
```

[4]:
```
# reading the csv file into a pandas dataframe
nyc=pd.read_csv('ave_hi_nyc_jan_1895-2018.csv')
```

[5]:
```
# displaying the first five rows of the dataframe
nyc.head()
```

[5]:

	Date	Value	Anomaly
0	189501	34.2	-3.2
1	189601	34.7	-2.7
2	189701	35.5	-1.9
3	189801	39.6	2.2
4	189901	36.4	-1.0

[6]:
```
# displaying the last five rows of the dataframe
nyc.tail()
```

[6]:

	Date	Value	Anomaly
119	201401	35.5	-1.9
120	201501	36.1	-1.3
121	201601	40.8	3.4
122	201701	42.8	5.4
123	201801	38.7	1.3

[7]:
```
# displaying random 20 rows of the dataframe
nyc.sample(20)
```

[7]:

	Date	Value	Anomaly
88	198301	38.0	0.6
21	191601	42.3	4.9

```
38   193301   45.6      8.2
36   193101   38.2      0.8
61   195601   36.2     -1.2
100  199501   42.6      5.2
4    189901   36.4     -1.0
106  200101   38.8      1.4
108  200301   32.9     -4.5
115  201001   38.1      0.7
94   198901   42.4      5.0
85   198001   38.0      0.6
80   197501   41.7      4.3
42   193701   45.7      8.3
29   192401   39.4      2.0
73   196801   31.7     -5.7
119  201401   35.5     -1.9
47   194201   36.5     -0.9
11   190601   42.3      4.9
66   196101   31.9     -5.5
```

[8]:
```
# declaring the column names of the dataframe
nyc.columns =['Date','Temperature','Anomaly']
```

[9]:
```
# displaying the first five rows of the dataframe
nyc.head()
```

[9]:
```
     Date  Temperature  Anomaly
0  189501         34.2     -3.2
1  189601         34.7     -2.7
2  189701         35.5     -1.9
3  189801         39.6      2.2
4  189901         36.4     -1.0
```

[10]:
```
# Takes the values of the Date column in the nyc dataframe and divides them by␣
  ↪100
nyc.Date=nyc.Date.floordiv(100)
```

[11]:
```
# displaying the first five rows of the dataframe
nyc.head()
```

[11]:
```
   Date  Temperature  Anomaly
0  1895         34.2     -3.2
1  1896         34.7     -2.7
2  1897         35.5     -1.9
3  1898         39.6      2.2
4  1899         36.4     -1.0
```

[12]:
```
# gets the dimensions of the nyc dataframe
nyc.shape
```

[12]: (124, 3)

[13]:
```
# displays the last five rows of the nyc dataframe
nyc.tail()
```

[13]:

	Date	Temperature	Anomaly
119	2014	35.5	-1.9
120	2015	36.1	-1.3
121	2016	40.8	3.4
122	2017	42.8	5.4
123	2018	38.7	1.3

[15]:
```
# importing the train_test_split function from the sklearn.model_selection␣
↪library
from sklearn.model_selection import train_test_split
```

[16]:
```
# This code accesses the Date column of the nyc dataset and returns the shape␣
↪of the values in that column.
nyc.Date.values.shape
```

[16]: (124,)

[17]:
```
# splitting the nyc dataframe into training and testing sets. The training set␣
↪will be used to train the model and the testing set will be used to test the␣
↪model.
X_train,X_test,y_train,y_test=train_test_split(nyc.Date.values.reshape(-1,1),␣
↪nyc.Temperature.values,random_state=11)
```

[18]:
```
# This code accesses the shape of the X_train variable(dimension of the␣
↪training set)
X_train.shape
```

[18]: (93, 1)

[19]:
```
# This code accesses the shape of the X_test variable(dimension of the testing␣
↪set)
X_test.shape
```

[19]: (31, 1)

[20]:
```
93+31
```

[20]: 124

[21]:
```
93/124*100
```

[21]: 75.0

[22]:
```
#Training the model
```

[23]:
```
# importing the LinearRegression class from the sklearn.linear_model library
from sklearn.linear_model import LinearRegression
```

[24]:
```
# creating an instance of the LinearRegression class
linear_regression=LinearRegression()
```

[25]:
```
# calling the linear_regression object instance.
linear_regression
```

[25]: LinearRegression()

[26]:
```
# training the model using the fit method of the linear_regression object␣
↪instance
linear_regression.fit(X=X_train, y=y_train)
```

[26]: LinearRegression()

[27]:
```
#Equation M and C
```

[28]:
```
# This code accesses the slope of the linear regression model(coefficient of␣
↪the linear regression model)
linear_regression.coef_
```

[28]: array([0.01939167])

[29]:
```
# intercept of the linear regression equation
linear_regression.intercept_
```

[29]: -0.30779820252656265

The equation: $Temperature = 0.01939167 * Date - 0.30779820252656265$

[30]:
```
# Testing the model
(0.01939167 * 2014) - 0.30779820252656265
```

[30]: 38.74702517747344

[31]:
```
# Testing the Model
```

[32]:
```
# This code accesses the predicted values of the linear regression model on␣
↪x_test
```

```
predicted = linear_regression.predict(X_test)
```

[33]:
```
# expected values of the linear regression model on the y_test set
expected = y_test
```

[34]:
```
# This code snippet is iterating over two lists, predicted and expected, and␣
 ↪printing out each corresponding pair of values.
for p,e in zip(predicted[::], expected[::]):
  print(f'Predicted: {p:.2f}, Expected: {e:.2f}')
```

```
Predicted: 37.86, Expected: 31.70
Predicted: 36.48, Expected: 35.50
Predicted: 37.93, Expected: 40.50
Predicted: 36.61, Expected: 29.80
Predicted: 36.75, Expected: 40.70
Predicted: 38.69, Expected: 34.80
Predicted: 36.44, Expected: 34.20
Predicted: 37.14, Expected: 38.20
Predicted: 37.62, Expected: 36.20
Predicted: 37.53, Expected: 42.50
Predicted: 37.00, Expected: 39.40
Predicted: 38.32, Expected: 40.90
Predicted: 37.20, Expected: 39.80
Predicted: 38.46, Expected: 40.80
Predicted: 36.56, Expected: 37.00
Predicted: 37.25, Expected: 45.70
Predicted: 38.18, Expected: 33.00
Predicted: 37.89, Expected: 29.90
Predicted: 38.15, Expected: 38.00
Predicted: 38.63, Expected: 42.40
Predicted: 38.05, Expected: 32.30
Predicted: 37.02, Expected: 33.80
Predicted: 37.12, Expected: 38.50
Predicted: 37.70, Expected: 37.80
Predicted: 36.73, Expected: 36.10
Predicted: 37.64, Expected: 33.80
Predicted: 37.56, Expected: 42.40
Predicted: 38.11, Expected: 30.60
Predicted: 36.87, Expected: 38.40
Predicted: 36.85, Expected: 42.30
Predicted: 36.94, Expected: 39.70
```

[1]:
```
# This code snippet is a for loop that iterates over two lists, predicted and␣
 ↪expected, and prints the predicted value, expected value, and the difference␣
 ↪between them (error) for each corresponding pair of values.
for p,e in zip(predicted[::], expected[::]):
  print(f'Predicted: {p:.2f}, Expected: {e:.2f}, Error: {e-p:.2f}')
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[1], line 2
      1 # This code snippet is a for loop that iterates over two lists, predicted and expected, and prints the predicted value, expected value, and the difference between them (error) for each corresponding pair of values. The zip() function is used to iterate over both lists simultaneously. The f-string is used to format the output string with the predicted value, expected value, and error. The :.2f format specifier is used to display the floating-point numbers with two decimal places.This code snippet is a for loop that iterates over two lists, predicted and expected, and prints the predicted value, expected value, and the difference between them (error) for each corresponding pair of values. The zip() function is used to iterate over both lists simultaneously. The f-string is used to format the output string with the predicted value, expected value, and error. The :.2f format specifier is used to display the floating-point numbers with two decimal places.
----> 2 for p,e in zip(predicted[::], expected[::]):
      3   print(f'Predicted: {p:.2f}, Expected: {e:.2f}, Error: {e-p:.2f}')

NameError: name 'predicted' is not defined
```

[36]:
```
# Mean Absolute Error (MAE)
```

[37]:
```
# importing the mean_absolute_error function from the sklearn.metrics library. Used to calculate the mean absolute error of the linear regression model
from sklearn.metrics import mean_absolute_error
```

[38]:
```
# printing the mean absolute error of the linear regression model
print("MAE", mean_absolute_error(expected,predicted))
```

```
MAE 3.450753706948741
```

[39]:
```
#RMSE - Root Mean Squared Error
```

[40]:
```
# importing the mean_squared_error function from the sklearn.metrics library. Used to calculate the mean squared error of the linear regression model
from sklearn.metrics import mean_squared_error
```

[41]:
```
# importing numpy as np for mathematical operations on arrays
import numpy as np
```

[42]:
```
# printing the root mean squared error of the linear regression model
print(f'RMSE',np.log(np.sqrt(mean_squared_error(expected, predicted)) ))
```

```
RMSE 1.4256936366057185
```

[43]:
```
# R Squared (R2)
```

[44]:
```
# importing the r2_score function from the sklearn.metrics library. Used to␣
 ↪calculate the r2 score of the linear regression model
from sklearn.metrics import r2_score
```

[45]:
```
# r2 variable stores the r2 score function of the linear regression model
r2=r2_score(expected,predicted)
```

[46]:
```
r2
```

[46]:
```
-0.033370346388810423
```

[47]:
```
# predict future model
```

[48]:
```
# The code you provided defines a lambda function named predict.
# The lambda function takes a single argument, x, and returns the result of the␣
 ↪linear regression model's coef_ attribute multiplied by x plus the␣
 ↪intercept_ attribute.


predict=(lambda x: linear_regression.coef_ *x + linear_regression.intercept_)
```

[49]:
```
# This code snippet calls the predict function and passes in the value 2014.
predict(2014)
```

[49]:
```
array([38.74703181])
```

[50]:
```
# This code snippet calls the predict function and passes in the value 2022.
predict(2022)
```

[50]:
```
array([38.9021652])
```

[51]:
```
# This code snippet calls the predict function and passes in the value 1800.
predict(1800)
```

[51]:
```
array([34.59721373])
```

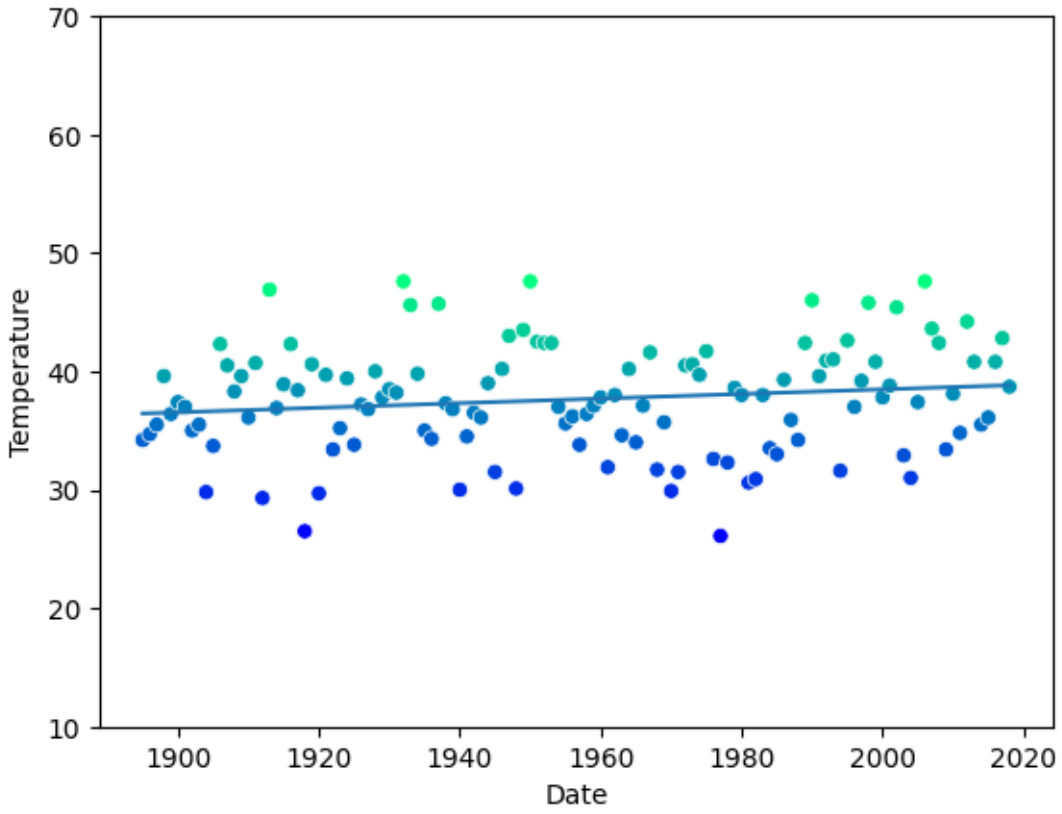
[52]:
```
# Visualizing the dataset with a Regression Line
```

[54]:
```
# importing the seaborn library as sns for data visualization
import seaborn as sns
```

```
Requirement already satisfied: seaborn in
/home/malims/Zen/myenv/lib/python3.11/site-packages (0.13.0)
Requirement already satisfied: numpy!=1.24.0,>=1.20 in
/home/malims/Zen/myenv/lib/python3.11/site-packages (from seaborn) (1.25.2)
Requirement already satisfied: pandas>=1.2 in
/home/malims/Zen/myenv/lib/python3.11/site-packages (from seaborn) (2.0.3)
```

```
Requirement already satisfied: matplotlib!=3.6.1,>=3.3 in
/home/malims/Zen/myenv/lib/python3.11/site-packages (from seaborn) (3.7.2)
Requirement already satisfied: contourpy>=1.0.1 in
/home/malims/Zen/myenv/lib/python3.11/site-packages (from
matplotlib!=3.6.1,>=3.3->seaborn) (1.1.0)
Requirement already satisfied: cycler>=0.10 in
/home/malims/Zen/myenv/lib/python3.11/site-packages (from
matplotlib!=3.6.1,>=3.3->seaborn) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in
/home/malims/Zen/myenv/lib/python3.11/site-packages (from
matplotlib!=3.6.1,>=3.3->seaborn) (4.41.1)
Requirement already satisfied: kiwisolver>=1.0.1 in
/home/malims/Zen/myenv/lib/python3.11/site-packages (from
matplotlib!=3.6.1,>=3.3->seaborn) (1.4.4)
Requirement already satisfied: packaging>=20.0 in
/home/malims/Zen/myenv/lib/python3.11/site-packages (from
matplotlib!=3.6.1,>=3.3->seaborn) (23.1)
Requirement already satisfied: pillow>=6.2.0 in
/home/malims/Zen/myenv/lib/python3.11/site-packages (from
matplotlib!=3.6.1,>=3.3->seaborn) (10.0.0)
Requirement already satisfied: pyparsing<3.1,>=2.3.1 in
/home/malims/Zen/myenv/lib/python3.11/site-packages (from
matplotlib!=3.6.1,>=3.3->seaborn) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in
/home/malims/Zen/myenv/lib/python3.11/site-packages (from
matplotlib!=3.6.1,>=3.3->seaborn) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in
/home/malims/Zen/myenv/lib/python3.11/site-packages (from pandas>=1.2->seaborn)
(2023.3)
Requirement already satisfied: tzdata>=2022.1 in
/home/malims/Zen/myenv/lib/python3.11/site-packages (from pandas>=1.2->seaborn)
(2023.3)
Requirement already satisfied: six>=1.5 in
/home/malims/Zen/myenv/lib/python3.11/site-packages (from python-
dateutil>=2.7->matplotlib!=3.6.1,>=3.3->seaborn) (1.16.0)
```

[55]:
```
# This code snippet creates a scatter plot of the nyc dataframe's Date and␣
  ↪Temperature columns.
axes=sns.scatterplot(data=nyc, x='Date',y='Temperature',
                     hue='Temperature', palette='winter', legend=False)
axes.set_ylim(10,70)
x=np.array([min(nyc.Date.values),max(nyc.Date.values)])
y=predict(x)
line=plt.plot(x,y)
```



[56]:
```
# The x variable stores the minimum and maximum values of the nyc dataframe's
 ↪Date column.
x
```

[56]: array([1895, 2018])

[57]:
```
# The y variable stores the predicted values of the linear regression model
 ↪based on the x variable values.
y
```

[57]: array([36.43942269, 38.82459851])

[ ]: