Data Pre-processing Assignment (12/Dec 2022) : Name : Ijaz Ullah

1: Mean Absolute Error(MAE)

Mean absolute error is a measure of errors between paired observations expressing the same phenomenon The Mean absolute error is calculated by adding up all the absolute errors and dividing them by the number of errors.

MAE (Mean absolute error) represents the difference between the original and predicted values extracted by averaged the absolute difference over the data set

```
# import the library
from sklearn.metrics import mean_absolute_error as MEA

 #actual and calculated numbers
actual = [13, 15, 15, 19]
calculated = [13, 13, 18, 17]

# calculate MAE
error = MEA(actual, calculated)

# display
print("Mean absolute error : " +str(error))
```

```
Mean absolute error : 1.75
```

2: Mean Squared Error(MSE)

Mean Squared Error evaluates the proximity of a regression line to a group of data points.

Mean squared error (MSE) is a measure of the error in prediction algorithms. This statistic quantifies the average squared variance between observed and predicted values. When there are no errors in a model, the MSE equals 0

MSE (Mean Squared Error) represents the difference between the original and predicted values extracted by squared the average difference over the data set.

It is an estimator measures the average of error squares

```
from sklearn.metrics import mean_squared_error


# Given values
Y_true = [1,1,2,2,4] # Y_true = Y (original values)

# calculated values
Y_pred = [1.10,1.29,1.99,2.69,3.4] # Y_pred = Y'

# Calculation of Mean Squared Error (MSE)
mean_squared_error(Y_true,Y_pred)
```

```
0.18606
```

3.Root mean squared error (RMSE)

RMSE (Root Mean Squared Error) is the error rate by the square root of MSE.

RSME (Root mean square error) calculates the transformation between values predicted by a model and actual values

Using RSME, we can easily measure the efficiency of the model.

RMSE is the square root of MSE. In case of unbiased estimator, RMSE is just the square root of variance, which is actually Standard Deviation

```
from sklearn.metrics import mean_squared_error

import math
actual= [34, 37, 44, 47, 48, 48, 46, 43, 32, 27, 26, 24]
pred = [37, 40, 46, 44, 46, 50, 45, 44, 34, 30, 22, 23]


# Calculation of Mean Squared Error (MSE)
MSE = mean_squared_error(actual, pred)

# Taking suare of  Mean Squared Error (MSE)
RMSE = math.sqrt(MSE)
print("Root Mean Square Error   Is:\n")
print(RMSE)

Root Mean Square Error   Is:

2.4324199198877374
```

4.Root Mean Squared Logarithmic Error (RMSLE)

RMSLE metric only considers the relative error between Predicted and the actual value and the scale of the error is not significant. On the other hand, RMSE value Increases in magnitude if the scale of error increases.

RMSLE is more precise than RMSE for data with larger variance.

RMSLE measures the ratio between actual and predicted.

RMSLE, you take the log of the predictions and actual values. So basically, what changes is the variance that you are measuring.RMSLE is usually used when you don't want to penalize huge differences in the predicted and the actual values when both predicted and true values are huge numbers.

If both predicted and actual values are small: RMSE and RMSLE is same.

If either predicted or the actual value is big: RMSE > RMSLE

If both predicted and actual values are big: RMSE > RMSLE (RMSLE becomes almost negligible)

```
import numpy as np
import sklearn.metrics as metrics
from sklearn.metrics import mean_squared_log_error

actual = np.array([56,45,68,49,26,40,52,38,30,48])
predicted = np.array([58,42,65,47,29,46,50,33,31,47])

print(mean_squared_log_error( actual, predicted, squared=False))
```

```
0.07801965746150723
```

5. R-squared

R-squared (Coefficient of determination) represents the coefficient of how well the values fit compared to the original values. The value from 0 to 1 interpreted as percentages. The higher the value is, the better the model is.

It measures the proportion of the variation in your dependent variable explained by all of your independent variables in the model

```
import numpy as np
import sklearn.metrics as metrics

y = np.array([-3, -1, -2, 1, -1, 1, 2, 1, 3, 4, 3, 5])
Y_pred = np.array([-2, 1, -1, 0, -1, 1, 2, 2, 3, 3, 3, 5])

# R sugared
r2  = metrics.r2_score(y,Y_pred)
r2
```

```
0.8655043586550436
```

6. Adjusted R square

It measures the proportion of variation explained by only those independent variables that really help in explaining the dependent variable. It penalizes you for adding independent variable that do not help in predicting the dependent variable.

Adjusted R-Squared can be calculated mathematically in terms of sum of squares. The only difference between R-square and Adjusted R-square equation is degree of freedom

Both R2 and the adjusted R2 give you an idea of how many data points fall within the line of the regression equation. However, there is one main difference between R2 and the adjusted R2: R2 assumes that every single variable explains the variation in the dependent variable. The adjusted R2 tells you the percentage of variation explained by only the independent variables that actually affect the dependent variable.

```
import numpy as np
import sklearn.metrics as metrics
```

```
actual = np.array([56,45,68,49,26,40,52,38,30,48])
predicted = np.array([58,42,65,47,29,46,50,33,31,47])

#calculate r-squared
r2_sk = metrics.r2_score(actual,predicted)
N=actual.shape[0]
p=3
x = (1-r2)
y = (N-1) / (N-p-1)
adj_rsquared = (1 - (x * y))
print("Adjusted-R2 : " , adj_rsquared)
```

```
Adjusted-R2 :  0.7982565379825655
```

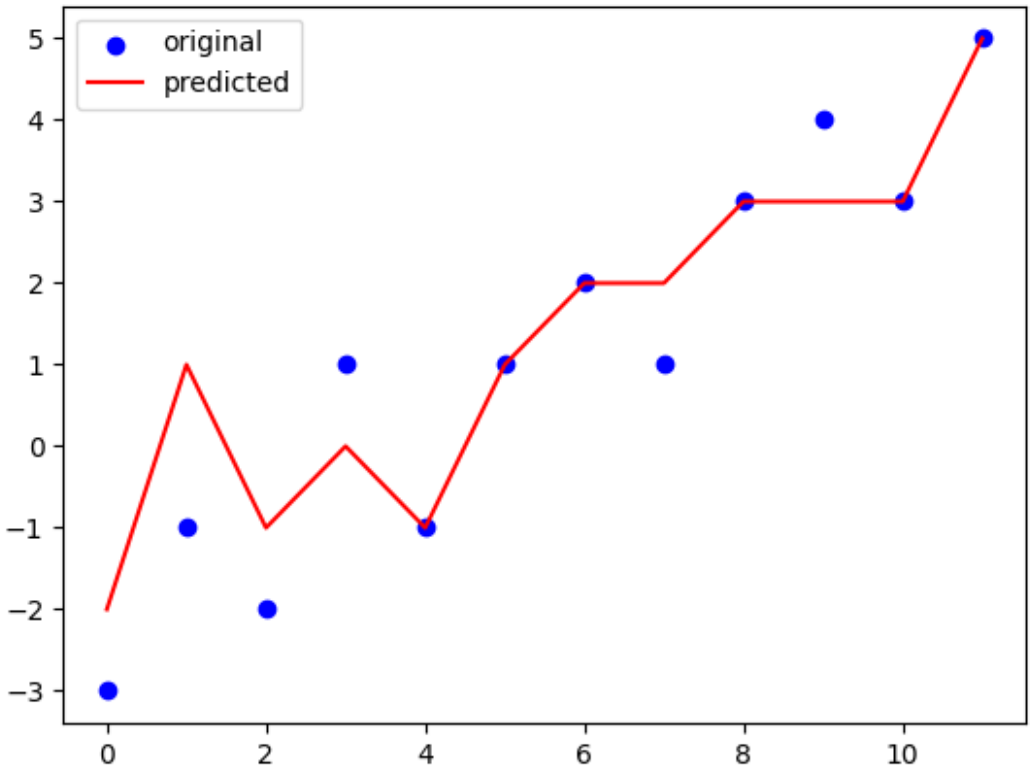
```
      Regression Accuracy Check in Python (MAE, MSE, RMSE, R-Squared)
```

```
import numpy as np
import sklearn.metrics as metrics
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_log_error

y = np.array([-3, -1, -2, 1, -1, 1, 2, 1, 3, 4, 3, 5])
Y_pred = np.array([-2, 1, -1, 0, -1, 1, 2, 2, 3, 3, 3, 5])
x = list(range(len(y)))
```

We can visualize them in a plot to check the difference visually.

```
plt.scatter(x, y, color="blue", label="original")
plt.plot(x, Y_pred, color="red", label="predicted")
plt.legend()
plt.show()
```




```
mae = metrics.mean_absolute_error(y, Y_pred)
mse = metrics.mean_squared_error(y, Y_pred)
rmse = np.sqrt(mse) # or mse**(0.5)
r2 = metrics.r2_score(y,Y_pred)

print("Results of sklearn.metrics:")
print("MAE:",mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R-Squared:", r2)
```

```
Results of sklearn.metrics:
MAE: 0.5833333333333334
MSE: 0.75
RMSE: 0.8660254037844386
R-Squared: 0.8655043586550436
```

Assignment (12/ Dec 2022) Preprocessing Categorical Data

Categorical data is a type of data that can be stored into groups or categories with the aid of names or labels. This grouping is usually made according to the data characteristics and similarities of these characteristics through a method known as matching

there are two kinds of categorical data-

Ordinal Data: The categories have an inherent order

Nominal Data: The categories do not have an inherent order

Label Encoding or Ordinal Encoding

This type of encoding is used when the variables in the data are ordinal, ordinal encoding converts each label into integer values and the encoded data represents the sequence of labels

In Label encoding, each label is converted into an integer value.

```
from sklearn.preprocessing import OrdinalEncoder

encoding = OrdinalEncoder()

X = [['Male', 1], ['Female', 3], ['Female', 2]]

#Fit the OrdinalEncoder to X.
encoding.fit(X)

OrdinalEncoder()

encoding.categories_

[array(['Female', 'Male'], dtype=object), array([1, 2, 3],
dtype=object)]

#    Transform X to ordinal codes.
encoding.transform([['Female', 3], ['Male', 1]])

array([[0., 2.],
       [1., 0.]])
```

2. One Hot Encoding

We use this categorical data encoding technique when the features are nominal(do not have any order). In one hot encoding, for each level of a categorical feature, we create a new variable. Each category is mapped with a binary variable containing either 0 or 1. Here, 0 represents the absence, and 1 represents the presence of that category.

The goal of one-hot encoding is to transform data from a categorical representation to a numeric representation.

```
import pandas as pd
from sklearn.preprocessing import LabelBinarizer
from sklearn.preprocessing import OneHotEncoder

ids = [11, 22, 33, 44, 55, 66, 77]
countries = ['Spain', 'France', 'Spain', 'Germany', 'France']

df = pd.DataFrame(list(zip(ids, countries)),
                  columns=['Ids', 'Countries'])
```

```
df
```

```
   Ids Countries
0   11     Spain
1   22    France
2   33     Spain
3   44   Germany
4   55    France
```

```
#antiate the LabelBinarizer and fit it:
y = LabelBinarizer().fit_transform(df.Countries)
y
```

```
array([[0, 0, 1],
       [1, 0, 0],
       [0, 0, 1],
       [0, 1, 0],
       [1, 0, 0]])
```

```
#let's populate a list and fit it in the encoder:
x = [[11, "Spain"], [22, "France"], [33, "Spain"], [44, "Germany"],
[55, "France"]]
```

```
y = OneHotEncoder().fit_transform(x).toarray()
print(y)
```

```
[[1. 0. 0. 0. 0. 0. 0. 1.]
 [0. 1. 0. 0. 0. 1. 0. 0.]
 [0. 0. 1. 0. 0. 0. 0. 1.]
 [0. 0. 0. 1. 0. 0. 1. 0.]
 [0. 0. 0. 0. 1. 1. 0. 0.]]
```

3.Dummy Encoding

Dummy coding scheme is similar to one-hot encoding. This categorical data encoding method transforms the categorical variable into a set of binary variables (also known as dummy variables). In the case of one-hot encoding, for N categories in a variable, it uses N binary variables. The dummy encoding is a small improvement over one-hot-encoding. Dummy encoding uses N-1 features to represent N labels/categories.

```
import category_encoders as ce
import pandas as pd
df = pd.DataFrame({'A': ['a', 'b', 'a'], 'B': ['b', 'a', 'c'],
                   'C': [1, 2, 3]})
#Original Data
df
```

```
   A  B  C
0  a  b  1
1  b  a  2
2  a  c  3
```

```
#encode the data
data_encoded=pd.get_dummies(data=df,drop_first=True)
data_encoded
```

```
   C  A_b  B_b  B_c
0  1    0    1    0
1  2    1    0    0
2  3    0    0    1
```

4.Effect Encoding

This encoding technique is also known as Deviation Encoding or Sum Encoding. Effect encoding is almost similar to dummy encoding, with a little difference. In dummy coding, we use 0 and 1 to represent the data but in effect encoding, we use three values i.e. 1,0, and -1.

```
import category_encoders as ce
import pandas as pd
data=pd.DataFrame({'City':
['Delhi','Mumbai','Hyderabad','Chennai','Bangalore','Delhi','Hyderabad
']})
encoder=ce.sum_coding.SumEncoder(cols='City',verbose=False,)

#Original Data
data
```

```
        City
0      Delhi
1     Mumbai
2  Hyderabad
3    Chennai
4  Bangalore
5      Delhi
6  Hyderabad
```

```
encoder.fit_transform(data)
```

```
c:\Users\Lenovo\.conda\envs\jazz\lib\site-packages\category_encoders\
base_contrast_encoder.py:126: FutureWarning: Intercept column might
not be added anymore in future releases (c.f. issue #370)
  warnings.warn("Intercept column might not be added anymore in future
releases (c.f. issue #370)",
c:\Users\Lenovo\.conda\envs\jazz\lib\site-packages\category_encoders\
base_contrast_encoder.py:126: FutureWarning: Intercept column might
not be added anymore in future releases (c.f. issue #370)
  warnings.warn("Intercept column might not be added anymore in future
releases (c.f. issue #370)",
```

```
   intercept  City_0  City_1  City_2  City_3
0          1     1.0     0.0     0.0     0.0
1          1     0.0     1.0     0.0     0.0
```

```
2          1     0.0     0.0     1.0     0.0
3          1     0.0     0.0     0.0     1.0
4          1    -1.0    -1.0    -1.0    -1.0
5          1     1.0     0.0     0.0     0.0
6          1     0.0     0.0     1.0     0.0
```

5.Binary Encoding

Binary encoding for categorical variables, similar to onehot, but stores categories as binary bitstrings.

```
import category_encoders as ce
```

```
import pandas as pd
df = pd.DataFrame({'A': ['a', 'b', 'a'], 'B': ['b', 'a', 'c']})
#Original Data
df
```

```
   A  B
0  a  b
1  b  a
2  a  c
```

```
X_trans = ce.BinaryEncoder().fit_transform(df)
X_trans.head()
```

```
   A_0  A_1  B_0  B_1
0    0    1    0    1
1    1    0    1    0
2    0    1    1    1
```

6.BaseN Encoding

In a positional number system, base or radix is the number of unique digits including zero used to represent numbers. In base n encoding if the base is two then the encoder will convert categories into the numerical form using their respective binary form which is formally one-hot encoding

```
import category_encoders as ce
import pandas as pd
```

```
data=pd.DataFrame({'Month':['January','April','March','April',
'Februay','June','July','June','September']})
```

```
data
```

```
       Month
0    January
1      April
2      March
3      April
4    Februay
```

```
5       June
6       July
7       June
8  September
```

```
#Create an object for Base N Encoding
encoder= ce.BaseNEncoder(cols=['Month'],return_df=True,base=5)

#Fit and Transform Data
data_encoded=encoder.fit_transform(data)

data_encoded
```

```
   Month_0  Month_1
0        0        1
1        0        2
2        0        3
3        0        2
4        0        4
5        1        0
6        1        1
7        1        0
8        1        2
```

7.Hash Encoding

Just like One-Hot encoding, the hash encoder converts the category into binary numbers using new data variables but here we can fix the number of new data variables

```
data=pd.DataFrame({'Month':['January','April','March','April',
'Februay','June','July','June','September']})

#Create object for hash encoder
encoder=ce.HashingEncoder(cols='Month',n_components=6)#Fit and
Transform Data
encoder.fit_transform(data)
```

```
   col_0  col_1  col_2  col_3  col_4  col_5
0      0      0      0      0      1      0
1      0      0      0      1      0      0
2      0      0      0      0      1      0
3      0      0      0      1      0      0
4      0      0      0      1      0      0
5      0      1      0      0      0      0
6      1      0      0      0      0      0
7      0      1      0      0      0      0
8      0      0      0      0      1      0
```

8.Target Encoding

Target encoding is the method of converting a categorical value into the mean of the target variable. This type of encoding is a type of bayesian encoding method where bayesian encoders use target variables to encode the categorical value.

```
df=pd.DataFrame({'name':[
'rahul','ashok','ankit','rahul','ashok','ankit'
],'marks' : [10,20,30,60,70,80,]})

df
```

```
    name  marks
0  rahul     10
1  ashok     20
2  ankit     30
3  rahul     60
4  ashok     70
5  ankit     80
```

```
#Create target encoding object
encoder=ce.TargetEncoder(cols='name')

#Fit and Transform Train Data
encoder.fit_transform(df['name'],df['marks'])
```

```
c:\Users\Lenovo\.conda\envs\jazz\lib\site-packages\category_encoders\
target_encoder.py:122: FutureWarning: Default parameter
min_samples_leaf will change in version 2.6.See
https://github.com/scikit-learn-contrib/category_encoders/issues/327
  warnings.warn("Default parameter min_samples_leaf will change in
version 2.6."
c:\Users\Lenovo\.conda\envs\jazz\lib\site-packages\category_encoders\
target_encoder.py:127: FutureWarning: Default parameter smoothing will
change in version 2.6.See
https://github.com/scikit-learn-contrib/category_encoders/issues/327
  warnings.warn("Default parameter smoothing will change in version
2.6."
```

```
        name
0  37.689414
1  45.000000
2  52.310586
3  37.689414
4  45.000000
5  52.310586
```