Temperature Forecasting …….

The data used in this problem is for the purpose of bias correction of next-day maximum and minimum air temperatures forecast of the LDAPS model operated by the Korea Meteorological Administration over Seoul, South Korea. This data consists of summer data from 2013 to 2017. The input data is largely composed of the LDAPS model's next-day forecast data, in-situ maximum and minimum temperatures of present-day, and geographic auxiliary variables. There are two outputs (i.e. next-day maximum and minimum air temperatures) in this data.

We need to observe the differences in the predicted values and actual values and increase the accuracy of our machine learning models.

The variables in the dataset are:

1. station - used weather station number: 1 to 25
2. Date - yyyy-mm-dd ('2013-06-30' to '2017-08-30')
3. Present_Tmax - Maximum air temperature between 0 and 21 h on the present day (Â°C): 20 to 37.6
4. Present_Tmin - Minimum air temperature between 0 and 21 h on the present day (Â°C): 11.3 to 29.9
5. LDAPS_RHmin - LDAPS model forecast of next-day minimum relative humidity (%): 19.8 to 98.5
6. LDAPS_RHmax - LDAPS model forecast of next-day maximum relative humidity (%): 58.9 to 100
7. LDAPS_Tmax_lapse - LDAPS model forecast of next-day maximum air temperature applied lapse rate (Â°C): 17.6 to 38.5
8. LDAPS_Tmin_lapse - LDAPS model forecast of next-day minimum air temperature applied lapse rate (Â°C): 14.3 to 29.6
9. LDAPS_WS - LDAPS model forecast of next-day average wind speed (m/s): 2.9 to 21.9
10. LDAPS_LH - LDAPS model forecast of next-day average latent heat flux (W/m2): -13.6 to 213.4
11. LDAPS_CC1 - LDAPS model forecast of next-day 1st 6-hour split average cloud cover (0-5 h) (%): 0 to 0.97
12. LDAPS_CC2 - LDAPS model forecast of next-day 2nd 6-hour split average cloud cover (6-11 h) (%): 0 to 0.97
13. LDAPS_CC3 - LDAPS model forecast of next-day 3rd 6-hour split average cloud cover (12-17 h) (%): 0 to 0.98
14. LDAPS_CC4 - LDAPS model forecast of next-day 4th 6-hour split average cloud cover (18-23 h) (%): 0 to 0.97
15. LDAPS_PPT1 - LDAPS model forecast of next-day 1st 6-hour split average precipitation (0-5 h) (%): 0 to 23.7
16. LDAPS_PPT2 - LDAPS model forecast of next-day 2nd 6-hour split average precipitation (6-11 h) (%): 0 to 21.6
17. LDAPS_PPT3 - LDAPS model forecast of next-day 3rd 6-hour split average precipitation (12-17 h) (%): 0 to 15.8
18. LDAPS_PPT4 - LDAPS model forecast of next-day 4th 6-hour split average precipitation (18-23 h) (%): 0 to 16.7

19. lat - Latitude (Â°): 37.456 to 37.645
20. lon - Longitude (Â°): 126.826 to 127.135
21. DEM - Elevation (m): 12.4 to 212.3
22. Slope - Slope (Â°): 0.1 to 5.2
23. Solar radiation - Daily incoming solar radiation (wh/m2): 4329.5 to 5992.9
24. Next_Tmax - The next-day maximum air temperature (Â°C): 17.4 to 38.9
25. Next_Tmin - The next-day minimum air temperature (Â°C): 11.3 to 29.8

The variable 'station' is the only categorical variable available in the data set.

The variable 'date' is an object type continuous variable.

Data Cleaning:

The data has to be cleaned and validated before building a model in order to make sure the model works pretty good and has minimal errors in it.

First import the data and copy it to a variable. Let it be named as df.

df.info() – Shows the count of non-null values in all the columns. We can notice if there are any missing values in any of the variables.

df.isnull().sum() – Shows the count of null values in each column

- ➔ The rows with null values can be removed, but the removal of data should be the last resort and the data loss should not exceed 8% as the best practice set to myself.
- ➔ To fill the null values, we can use the basic methods such as forward fill or backward fill. (df.fillna(method = 'ffill') , df.fillna(method='bfill'))

In this problem, the data loss if removed the null values is around 2%. Hence, I have dropped all the rows with null values.

```
df = df.dropna()
```

The Date column is object type and needs to be converted to numerical to be used in modelling. The Date column first is converted to datetime format and day of week, months and year data is extracted from it.

```
df['Date'] = pd.to_datetime(df['Date'],dayfirst = True)
```

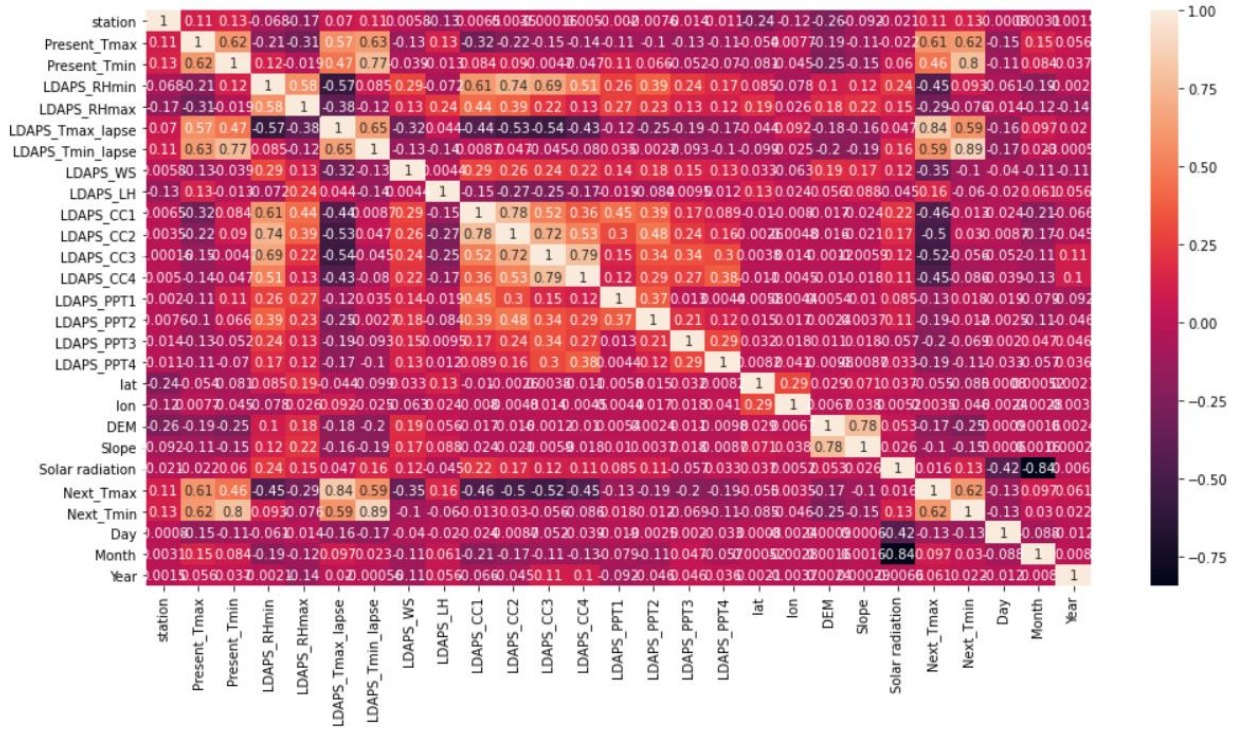
The null values are removed and object type data is converted, extracted and added to the data. Since, all the columns are in numerical type we can proceed further.

EDA:

The variables in this data are continuous apart from ‘station’. So, we can make use of distplot, boxplot, heatmap and so on..

Heatmap to show correlation between variables:

```
plt.figure(figsize = (16,8))
sns.heatmap(df.corr(),annot=True)
```



The Tmax, Tmin are having significant positive correlation (present, next, ldaps tmax and tmin). Month is highly negatively correlated with solar radiation. And it is obvious that, during the rainy season, solar radiation will be less. DEM and slope are also significantly correlated

The columns with correlation values of near to ‘0’ with the target variables have less impact on the prediction of target variable. Therefore, we can remove them using drop.

Now to check the distribution, we can use the below code:

```
sns.distplot(df[#column name for which distribution has to be checked] )
```

Since, there are many columns with continuous type of data I won’t be adding the graphs here. Please visit the below github link for the ipynb file:

https://github.com/vangetiakhil/Evaluation-Projects/blob/main/Temperature.ipynb

Since, the data is not normally distributed, there is a skewness which can be due to the outliers.

To check the skewness,

```
df.skew()
```

The columns with skewness between -0.5 to 0.5 are acceptable. This does not mean the data is normally distributed. We even need to see for that.

For checking outliers,

```
sns.boxplot(df[# column name])
```

link : https://github.com/vangetiakhil/Evaluation-Projects/blob/main/Temperature.ipynb

When trying to remove the outliers using zscore method,

```
from scipy.stats import zscore

z = np.abs(zscore(df))
threshold = 3
np.where(z>3)

df_new = df[(z<3).all(axis=1)]
```

The data loss is around 15% which is not acceptable, hence tried to use transformations on the columns with outliers. The transformations are log, sqrt, cbrt, boxcox. The transformation is specific to each column depending on the values in it.

For example :- log transformation can't be used on the column which has '0' as a value in it. So, choose according to the data.

In this problem, I have used log, cbrt transformations.

The variable 'station' is categorical and needs to be encoded. This can be done by the use of LabelEncoder and OneHotEncoder. The column has 25 unqiue values.

Instead of using LabelEncoder, I went with the use of dummies.

```
dummies = pd.get_dummies(df['station'])

df = pd.concat([df,dummies],axis=1)
```

Preprocessing:

To build a good model, all columns must be under same limits. Standardization is the option for that. We only need to standardize the independent variables. Therefore, splitting the data into target (y) and independent (x) data frames.

Now standardizing the X.

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

scaledX = pd.DataFrame(scaler.fit_transform(X),columns=X.columns)
```

The problem here is to predict the target variables Tmax and Tmin which are continuous type. Therefore, it is a regression problem.

Since, there are two target variables, few regression models won't work such as SVR (support vector regressor) etc;

We can proceed with linear regression, Random forest, lasso, ridge, Knearest neighbors.

The metrics we can use this problem are r2_score, mean squared_error, root mean squared error, mean absolute error. The best are r2 score and root mean squared error.

After building the models and finding their r2 score, check that with the cross val score. Prefer the model with less difference.

Tune the parameters for that modelling technique and build the final model.

For the detailed process, please do visit the link:

https://github.com/vangetiakhil/Evaluation-Projects/blob/main/Temperature.ipynb