Weather Data Prediction

❖ Introduction :

Weather profoundly influences agriculture, transportation, energy consumption, and day-to-day decision-making. Accurate forecasts give farmers the foresight to protect crops, help airlines optimize flight paths, and let city planners mitigate the impact of extreme events. Traditional numerical-weather-prediction (NWP) models—while physically rigorous—are computationally intensive and can struggle to capture fine-scale, nonlinear patterns hidden in large, high-frequency sensor streams.

In this project, we build a **data-centric weather-prediction pipeline** that leverages supervised ML techniques to forecast key meteorological variables such as temperature, humidity, and wind speed.

❖ Content :

-: Project Title :-

Weather Data Prediction

-: Scenario :-

- **Global Airline Weather Risk Management :**

A major international airline, SkyWings, operates hundreds of flights daily across continents. Weather disturbances like storms, fog, and strong winds often cause delays, diversions, and safety risks. Traditional weather forecasting systems provide general predictions, but they often lack localized accuracy and real-time adaptability across diverse regions.

To improve operations, SkyWings partners with a climate-tech company that uses machine learning models trained on worldwide weather datasets, including satellite data, radar images, and sensor readings from airports across the globe.

- **What the ML System Does :**

- Predicts **turbulence zones**, **thunderstorm developments**, and **wind shear** patterns With higher spatial and temporal precision.
- Uses deep learning models like **ConvLSTM and Transformers** to analyze real-time atmospheric data.
- Sends automatic alerts to air traffic control and pilots, allowing for **dynamic route adjustments**.

❖ Solutions :

To predict a model we have to follow this steps :

- Prepare Data
- Split Data
- Train Model
- Predict
- Evaluate: Compare predictions to true labels

```
:  import pandas as pd
   import numpy as np
   import matplotlib.pyplot as plt

   from sklearn.model_selection import train_test_split
   from sklearn.ensemble import RandomForestRegressor
   from sklearn.metrics import mean_absolute_error, r2_score
```

Import DataSet :-

```
In [2]: w_d = pd.read_csv( "C:\\Numpy Data Sets\\Weather Data\\weatherHistory.csv\\weatherHistory.csv" )
        w_d.head()
```

Out[2]:

	Formatted Date	Summary	Precip Type	Temperature (C)	Apparent Temperature (C)	Humidity	Wind Speed (km/h)	Wind Bearing (degrees)	Visibility (km)	Loud Cover	Pressure (millibars)	Daily Summary
0	2006-04-01 00:00:00.000 +0200	Partly Cloudy	rain	9.472222	7.388889	0.89	14.1197	251.0	15.8263	0.0	1015.13	Partly cloudy throughout the day.
1	2006-04-01 01:00:00.000 +0200	Partly Cloudy	rain	9.355556	7.227778	0.86	14.2646	259.0	15.8263	0.0	1015.63	Partly cloudy throughout the day.
2	2006-04-01 02:00:00.000 +0200	Mostly Cloudy	rain	9.377778	9.377778	0.89	3.9284	204.0	14.9569	0.0	1015.94	Partly cloudy throughout the day.
3	2006-04-01 03:00:00.000 +0200	Partly Cloudy	rain	8.288889	5.944444	0.83	14.1036	269.0	15.8263	0.0	1016.41	Partly cloudy throughout the day.
4	2006-04-01 04:00:00.000 +0200	Mostly Cloudy	rain	8.755556	6.977778	0.83	11.0446	259.0	15.8263	0.0	1016.51	Partly cloudy throughout the day.

Drop Columns :-

[Remove columns that are either unnecessary or unhelpful for a machine learning model, like text & datetime]

"Precip Type" is a useful categorical feature, that tells us what kind of precipitation occurred "rain", "snow" etc.

This kind of information is directly related to the weather and it impacts many target variables like :

[Temperature]
[Apparent temperature]
[Wind Speed]
[Visibility]

That's why I don't remove this column]

```
w_d = w_d.drop( columns = [ "Formatted Date", "Summary", "Daily Summary" ])

w_d.head()
```

	Precip Type	Temperature (C)	Apparent Temperature (C)	Humidity	Wind Speed (km/h)	Wind Bearing (degrees)	Visibility (km)	Loud Cover	Pressure (millibars)
0	rain	9.472222	7.388889	0.89	14.1197	251.0	15.8263	0.0	1015.13
1	rain	9.355556	7.227778	0.86	14.2646	259.0	15.8263	0.0	1015.63
2	rain	9.377778	9.377778	0.89	3.9284	204.0	14.9569	0.0	1015.94
3	rain	8.288889	5.944444	0.83	14.1036	269.0	15.8263	0.0	1016.41
4	rain	8.755556	6.977778	0.83	11.0446	259.0	15.8263	0.0	1016.51

Check Missing Values :-

[isnull() is used to find how many missing or NaN values are in each column of the DataFrame]

[sum() function in pandas is used to add all values in a Series in a DataFrame]

This counts how many NaN values are in each column :

[True is treated as 1]

[False is treated as 0]

```
w_d.isnull().sum()
```

```
Precip Type                517
Temperature (C)              0
Apparent Temperature (C)     0
Humidity                     0
Wind Speed (km/h)            0
Wind Bearing (degrees)       0
Visibility (km)              0
Loud Cover                   0
Pressure (millibars)         0
dtype: int64
```

Drop Rows With Missing or NaN Values :-

[It's used to remove rows with any missing or NaN values from the DataFrame w_d.]

-> It checks each row in the DataFrame.

-> If any value in the row is NaN (missing), that entire row is removed.

-> A new DataFrame is returned with only complete (non-null) rows.

-> Saves the cleaned version back to the same variable.

```
w_d = w_d.dropna()
w_d
```

	Precip Type	Temperature (C)	Apparent Temperature (C)	Humidity	Wind Speed (km/h)	Wind Bearing (degrees)	Visibility (km)	Loud Cover	Pressure (millibars)
0	rain	9.472222	7.388889	0.89	14.1197	251.0	15.8263	0.0	1015.13
1	rain	9.355556	7.227778	0.86	14.2646	259.0	15.8263	0.0	1015.63
2	rain	9.377778	9.377778	0.89	3.9284	204.0	14.9569	0.0	1015.94
3	rain	8.288889	5.944444	0.83	14.1036	269.0	15.8263	0.0	1016.41
4	rain	8.755556	6.977778	0.83	11.0446	259.0	15.8263	0.0	1016.51
...	...	...	...	...	...	...	...	...	...
96448	rain	26.016667	26.016667	0.43	10.9963	31.0	16.1000	0.0	1014.36
96449	rain	24.583333	24.583333	0.48	10.0947	20.0	15.5526	0.0	1015.16
96450	rain	22.038889	22.038889	0.56	8.9838	30.0	16.1000	0.0	1015.66
96451	rain	21.522222	21.522222	0.60	10.5294	20.0	16.1000	0.0	1015.95
96452	rain	20.438889	20.438889	0.61	5.8765	39.0	15.5204	0.0	1016.16

95936 rows × 9 columns

Encode Categorical Column :-

[This line converts the categorical column 'Precip Type' into numerical or binary columns using one-hot encoding]

[To convert a categorical column into binary columns, we get one new column for each category by default]

[Drop the first category column and keep only the rest. To avoid multicollinearity, especially in linear models]

```
w_d = pd.get_dummies( w_d, columns = [ "Precip Type" ] )
w_d


# [ Alternative ]

# w_d[ "Precip Type" ] = w_d[ "Precip Type" ].map({ "rain" : 0 "snow" : 1 })
```

	Temperature (C)	Apparent Temperature (C)	Humidity	Wind Speed (km/h)	Wind Bearing (degrees)	Visibility (km)	Loud Cover	Pressure (millibars)	Precip Type_rain	Precip Type_snow
0	9.472222	7.388889	0.89	14.1197	251.0	15.8263	0.0	1015.13	True	False
1	9.355556	7.227778	0.86	14.2646	259.0	15.8263	0.0	1015.63	True	False
2	9.377778	9.377778	0.89	3.9284	204.0	14.9569	0.0	1015.94	True	False
3	8.288889	5.944444	0.83	14.1036	269.0	15.8263	0.0	1016.41	True	False
4	8.755556	6.977778	0.83	11.0446	259.0	15.8263	0.0	1016.51	True	False
...	...	...	...	...	...	...	...	...	...	...
96448	26.016667	26.016667	0.43	10.9963	31.0	16.1000	0.0	1014.36	True	False
96449	24.583333	24.583333	0.48	10.0947	20.0	15.5526	0.0	1015.16	True	False
96450	22.038889	22.038889	0.56	8.9838	30.0	16.1000	0.0	1015.66	True	False
96451	21.522222	21.522222	0.60	10.5294	20.0	16.1000	0.0	1015.95	True	False
96452	20.438889	20.438889	0.61	5.8765	39.0	15.5204	0.0	1016.16	True	False

95936 rows × 10 columns

Define Features(X) and Target(y) :-

Splitting dataset into :

-> X = Input features. This is what we give to the model to help it learn.

-> y = Target variable. This is what we want the model to predict.

```
X = w_d.drop( columns = [ "Temperature (C)" ])  # [ Features Variable ]
y = w_d[ "Temperature (C)" ]                    # [ Target Variable ]
```

Split Into Train-Test :-

X_train : Training the model. 80% of X

y_train : Training labels. 80% of y

X_test : Testing the model. 20% of X

y_test : Testing labels. 20% of y

-> X_train and y_train : The model learns patterns from this data.

-> use X_test to make predictions.

-> You compare those predictions to y_test.

-> This tells how well the model performs on unseen data.

```
X_train, X_test, y_train, y_test = train_test_split( X,y, test_size = 0.2, random_state = 42 )
```

Train The Model :-

To train a machine learning model that can predict temperature based on weather conditions, we doing this.

If we try to predict without training, the model knows nothing — it hasn't seen any data and it will make random guesses.

```
Syntax : model.fit(X_train, y_train)

-> [ fit() function is used to train a machine learning model. It tells the model to learn from the training data ]
```

=> Suitable Models for Regression :

```
-> [ RandomForstRegressor ]
-> [ DecisionTreeRegressor ]
-> [ LinearRegresssion ]
```

Random Forest is a machine learning algorithm that uses many "decision trees" to make predictions.

It's part of a technique called ensemble learning, where multiple models are combined to get better results.

```
model = RandomForestRegressor()
model.fit( X_train, y_train )
```

▸ RandomForestRegressor

Evaluate the Model :-

=> mean_absolute_error : The "average" of the absolute differences between the "actual values" and the "predicted values".

```
-> 0 : Perfect prediction, no error at all.
-> > 0 : Some error or average difference between predicted and actual values.
-> ∞ : Extremely bad predictions or huge errors.
-> < 0 : Cannot be negative.
```

=> r2_score : R-squared or Coefficient of Determination. Tells you how well your model is performing. Means, how well the model's predictions fit the actual data.

```
-> If r2_score = 1 : The model predictions are perfect.
-> If r2_score = 0 : The model predictions are no better, just predicting the average.
-> If r2_score < 0 : The model is worse than guessing.
```

```
y_pred = model.predict( X_test )

print( "MAE :", mean_absolute_error( y_pred, y_test ))
print( "R2 Score :", r2_score( y_pred, y_test ))
```

```
MAE : 0.01267439719269674
R2 Score : 0.9999660789084093
```

Visualize The Predicting Model :-

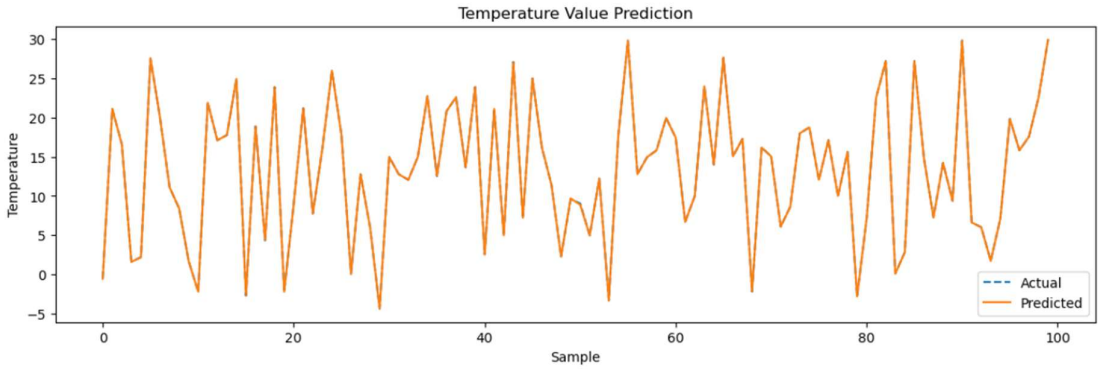
=> values : Converts the pandas Series to a NumPy array.

=> [:100] : Takes only the first 100 samples.

```
plt.figure( figsize = ( 14,4 ))

plt.plot( y_test.values[ :100 ], label = "Actual", linestyle = "--" )
plt.plot( y_pred[ :100 ], label = "Predicted" )

plt.title( "Temperature Value Prediction" )
plt.xlabel( "Sample" )
plt.ylabel( "Temperature" )
plt.legend()
plt.show()
```



❖ Acknowledgement :

I would like to express my sincere gratitude to everyone who supported and guided me throughout this machine learning–based weather prediction project.

First and foremost, I would like to thank my mentor **Debjit Saha** for their valuable advice, continuous support and encouragement during every stage of this work. Their insights into machine learning models and real-world applications have greatly enriched my understanding.