



Group assignment report: WeatherSense

Prepared For

Assoc. Prof.Dr.Kitsana Waiyamai

Prepared By

6510545748

6510545721

Sukprachoke

Wissarut

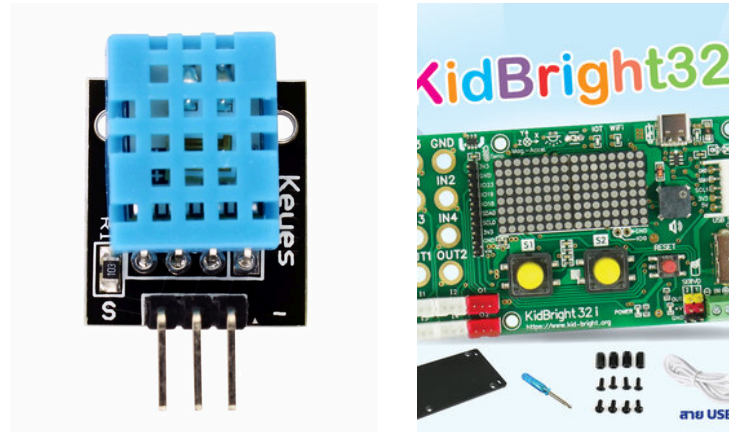
Leelapisuth (Dropped)

Kanasub

This Project is collected and evaluated in 01219367 Data Analytics
Software and knowledge Engineers

Primanny data

Our primary data is collected by the Kidbright board, incorporating a temperature and humidity sensor from the KY-015 module, and using MQTT to send a data to NodeRED

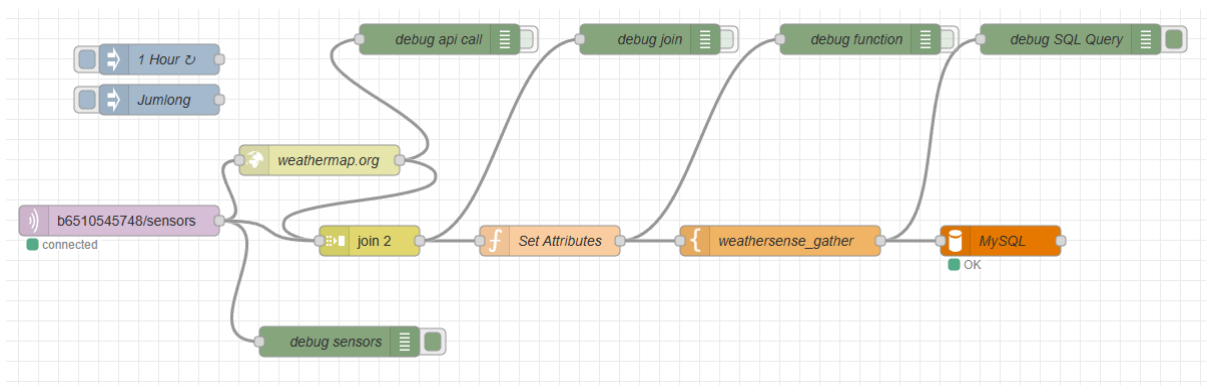


Secondary Data

Our secondary data is collected by call api from OpenWeatherMap API (Current weather) Using NodeRED to fetch data from API

attribute select:

- Humidity (%)
- Temperature (degree celsius)
- Pressure (hPa)
- Cloundiness (%)
- Weather: such as Cloud, Few clouds etc.



(NodeRED in use)

Merge primary data and secondary and upload to our database

Our database table structure below:

Table structure

Relation view

| | # | Name | Type | Collation | Attributes | Null | Default | Comments | Extra | Action |
|--------------------------|----|-----------------|--------------|--------------------|------------|------|-------------------|----------|-------------------|--------------------|
| <input type="checkbox"/> | 1 | id | int | | | No | None | | AUTO_INCREMENT | Change Drop More |
| <input type="checkbox"/> | 2 | ts | timestamp | | | No | CURRENT_TIMESTAMP | | DEFAULT_GENERATED | Change Drop More |
| <input type="checkbox"/> | 3 | temp_sensor | float | | | No | None | | | Change Drop More |
| <input type="checkbox"/> | 4 | humidity_sensor | float | | | No | None | | | Change Drop More |
| <input type="checkbox"/> | 5 | temp_api | float | | | No | None | | | Change Drop More |
| <input type="checkbox"/> | 6 | humidity_api | float | | | No | None | | | Change Drop More |
| <input type="checkbox"/> | 7 | pressure | float | | | No | None | | | Change Drop More |
| <input type="checkbox"/> | 8 | wind_speed | float | | | No | None | | | Change Drop More |
| <input type="checkbox"/> | 9 | cloudiness | float | | | No | None | | | Change Drop More |
| <input type="checkbox"/> | 10 | weather | varchar(255) | utf8mb3_general_ci | | No | None | | | Change Drop More |

☐ Check all

With selected:

Browse

Change

Drop

Primary

Unique

Index

Spatial

Fulltext

All attributes will be used

Data Exploration

Using python Pandas to explore and preprocessing more over training a model.

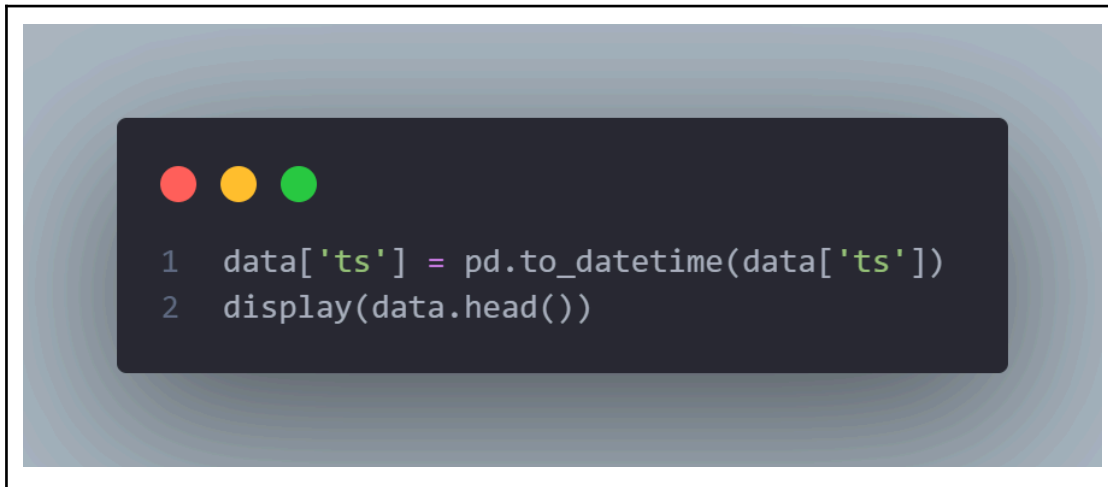
- Checking data type of dataframe

```

id          int64
ts          object
temp_sensor int64
humidity_sensor int64
temp_api    float64
humidity_api int64
pressure    int64
wind_speed  float64
cloudiness  int64
weather     object
dtype: object

```

- Then we need to change type of ts to DateTime type

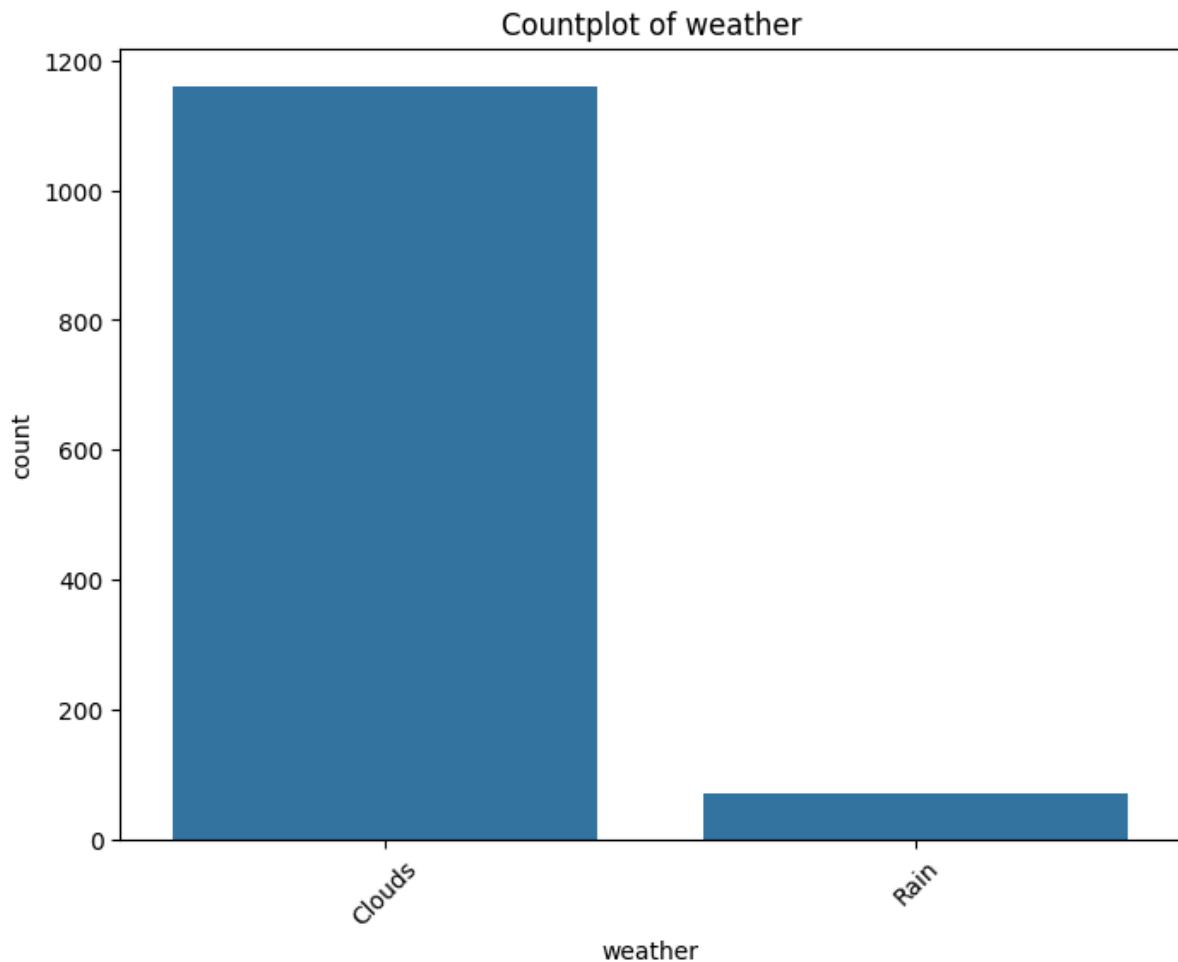


```
1 data['ts'] = pd.to_datetime(data['ts'])
2 display(data.head())
```

- Retrieve a summary statistic of this dataset.

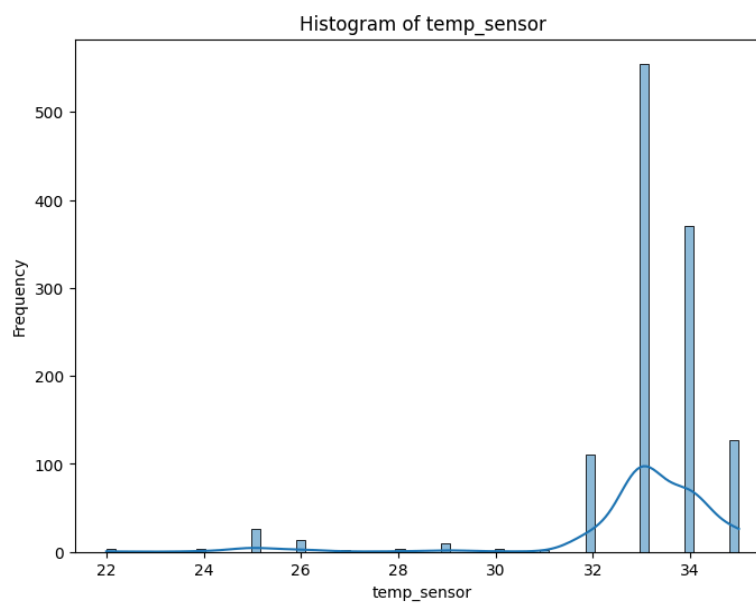
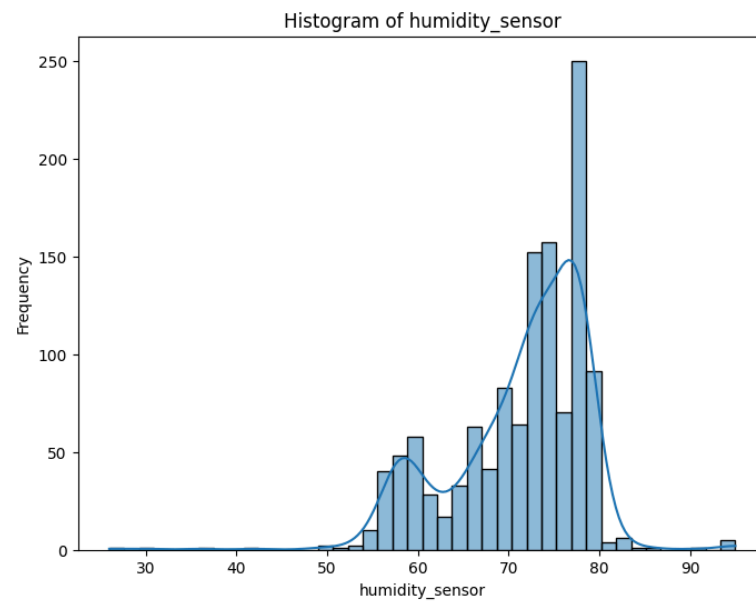
| | id | ts | temp_sensor | humidity_sensor | temp_api | humidity_api | pressure | wind_speed | cloudiness |
|-------|-------------|-------------------------------|-------------|-----------------|-------------|--------------|-------------|-------------|-------------|
| count | 1232.000000 | 1232 | 1232.000000 | 1232.000000 | 1232.000000 | 1232.000000 | 1232.000000 | 1232.000000 | 1232.000000 |
| mean | 616.500000 | 2024-04-28 05:42:56.026785792 | 33.039773 | 71.299513 | 33.911964 | 64.646916 | 1005.815747 | 4.595649 | 21.363636 |
| min | 1.000000 | 2024-04-21 17:34:59 | 22.000000 | 26.000000 | 28.860000 | 26.000000 | 1000.000000 | 1.030000 | 20.000000 |
| 25% | 308.750000 | 2024-04-24 20:33:47.750000128 | 33.000000 | 68.000000 | 31.130000 | 52.000000 | 1005.000000 | 3.600000 | 20.000000 |
| 50% | 616.500000 | 2024-04-27 14:51:34 | 33.000000 | 73.000000 | 32.690000 | 70.000000 | 1006.000000 | 4.630000 | 20.000000 |
| 75% | 924.250000 | 2024-05-01 18:49:37.500000 | 34.000000 | 77.000000 | 37.190000 | 78.000000 | 1007.000000 | 5.140000 | 20.000000 |
| max | 1232.000000 | 2024-05-07 02:21:56 | 35.000000 | 95.000000 | 41.220000 | 90.000000 | 1010.000000 | 8.230000 | 40.000000 |
| std | 355.792074 | NaN | 1.904941 | 7.383676 | 3.152121 | 15.168092 | 1.842096 | 1.232721 | 5.043200 |

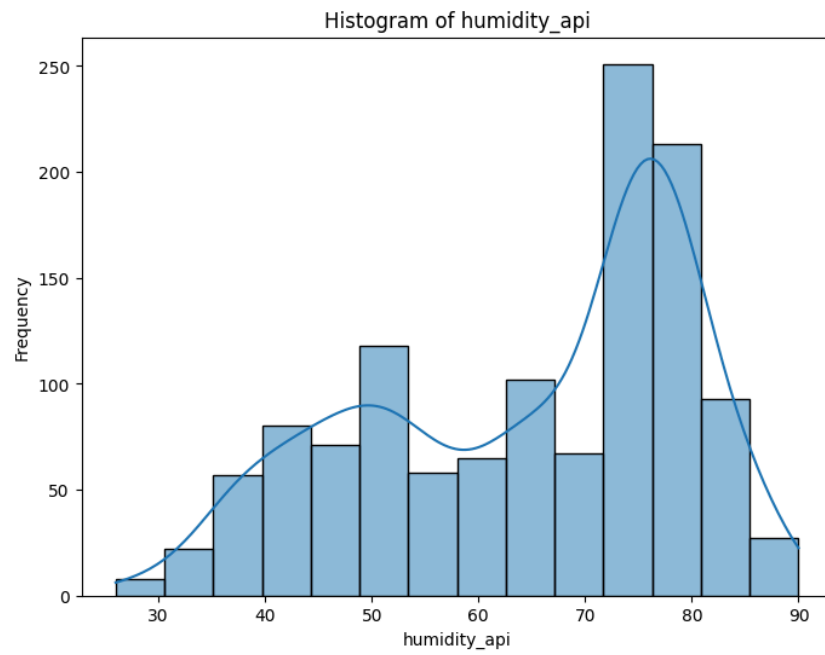
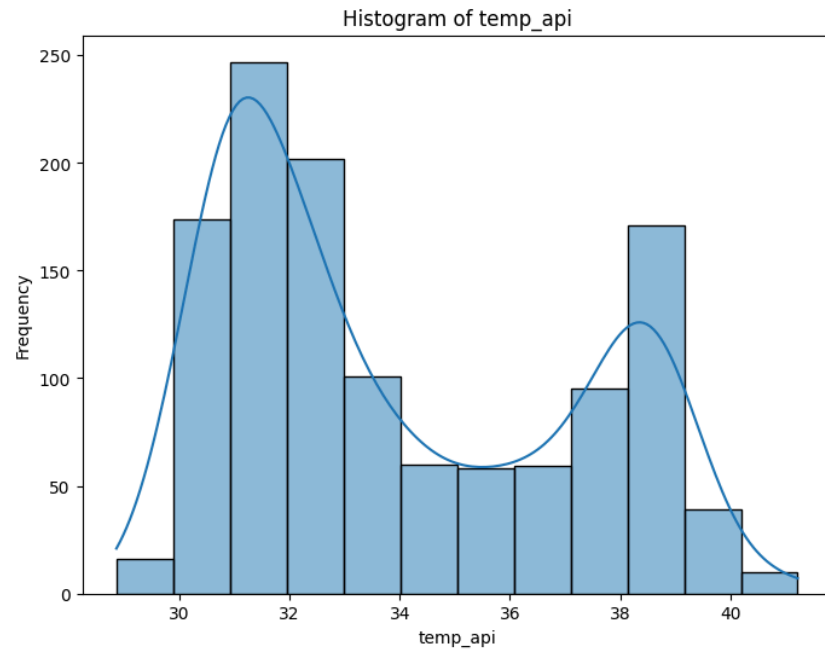
- For categorical features using count plots.

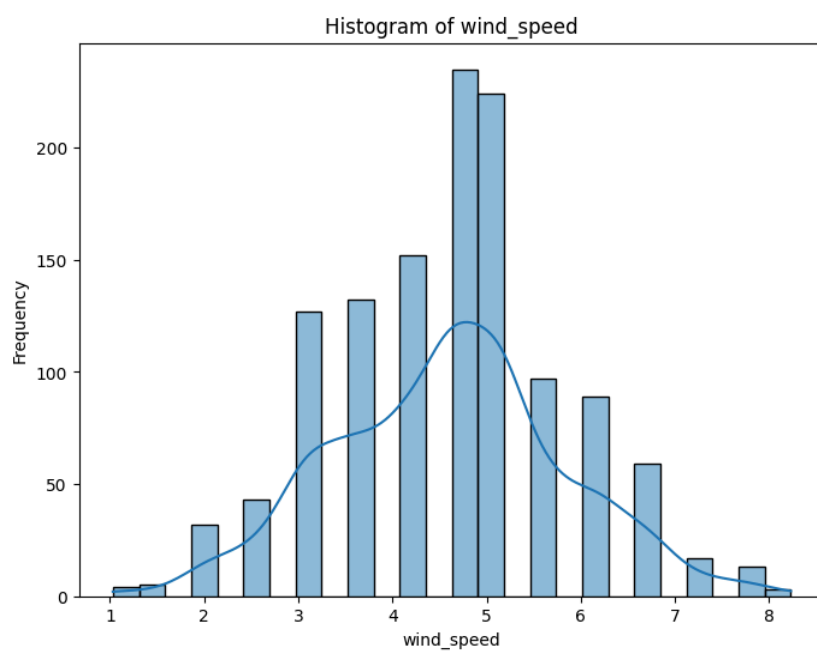
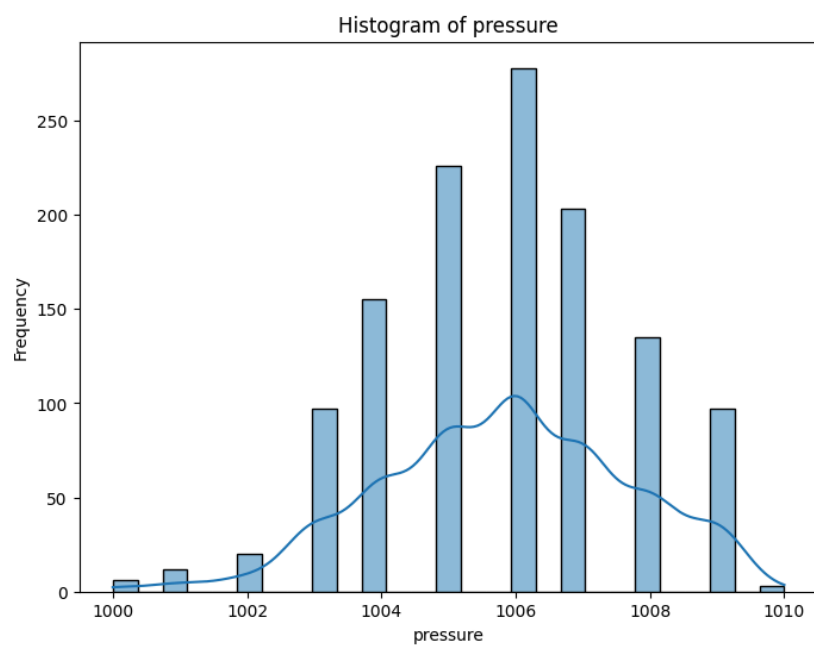


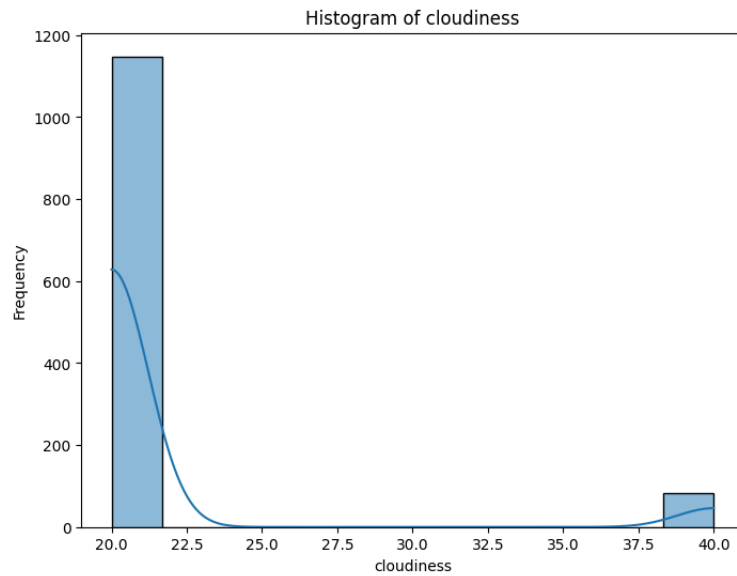
From the histogram plot, it's evident that the data clouds contain a large number of values, whereas instances of rain are scarce. This imbalance can lead to a data problem, which we need to address during the preprocessing stage.

- For numerical features using histogram plot to find a distribution.

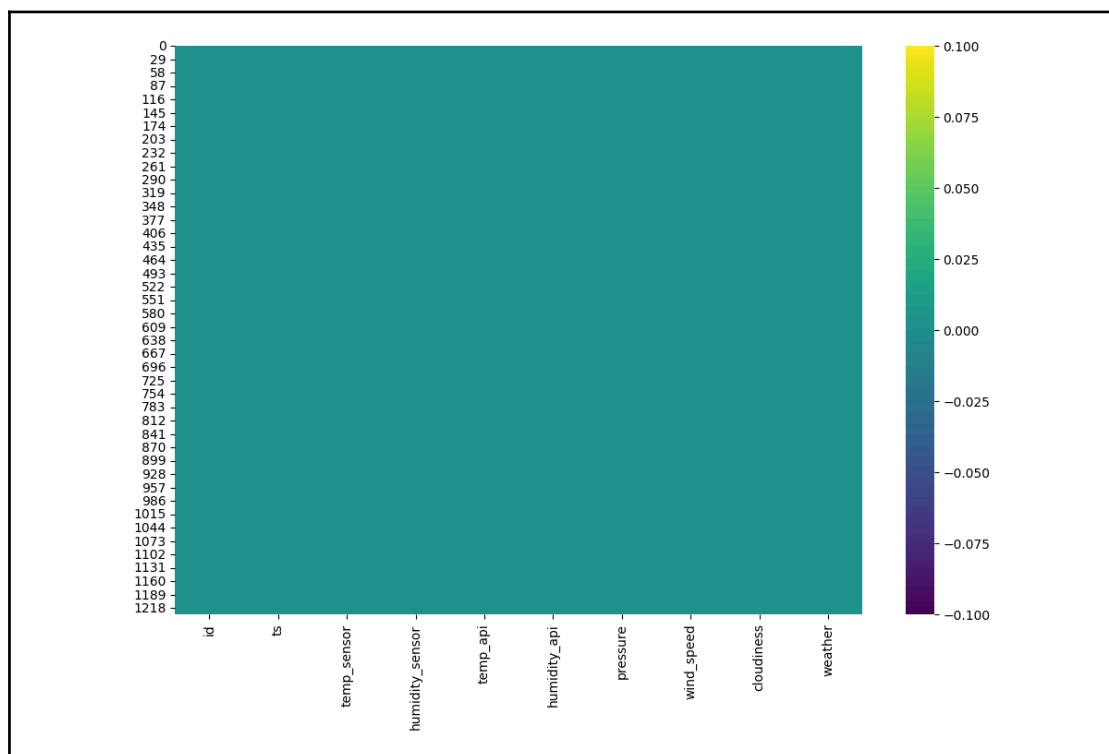




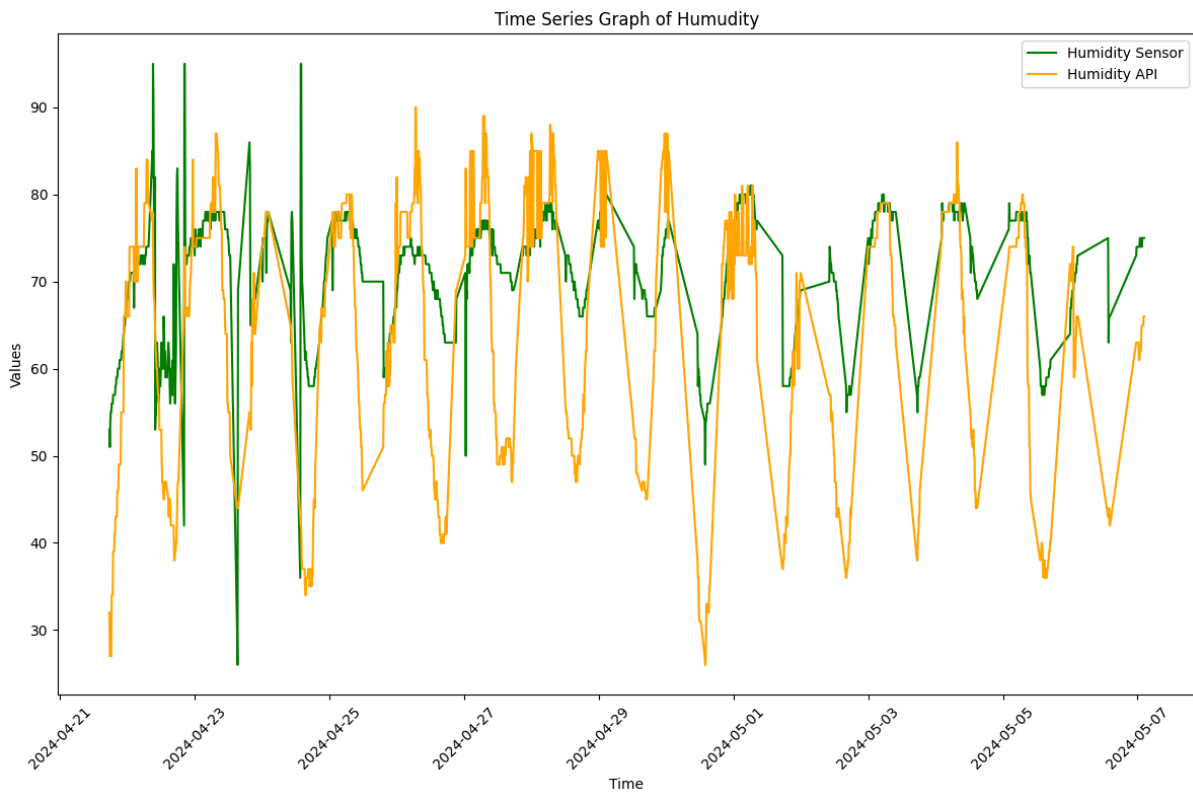
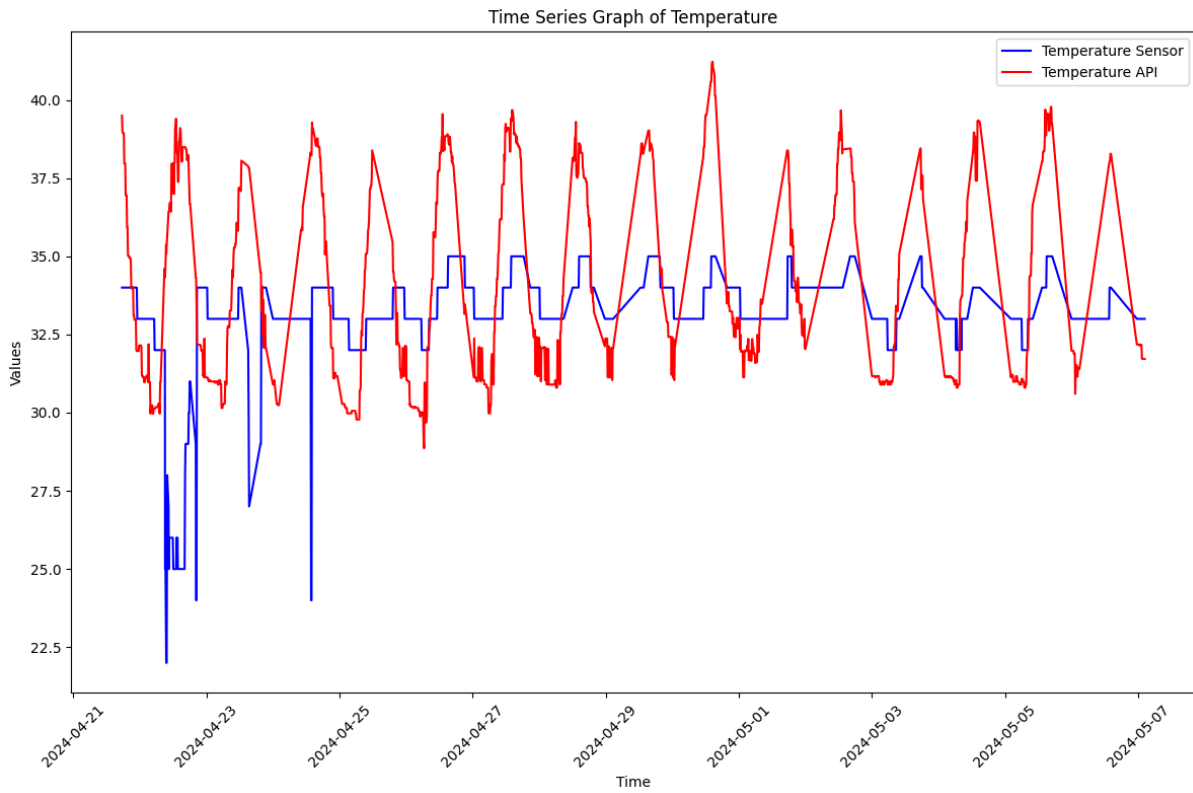


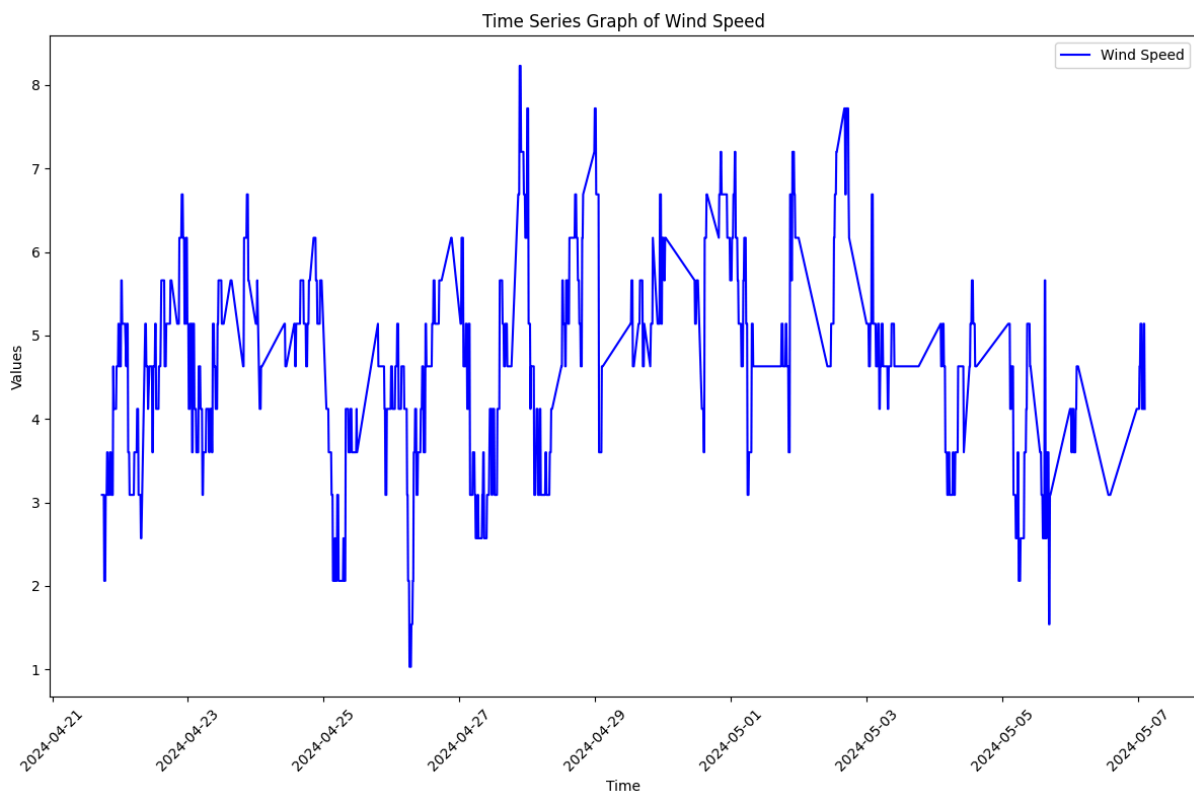
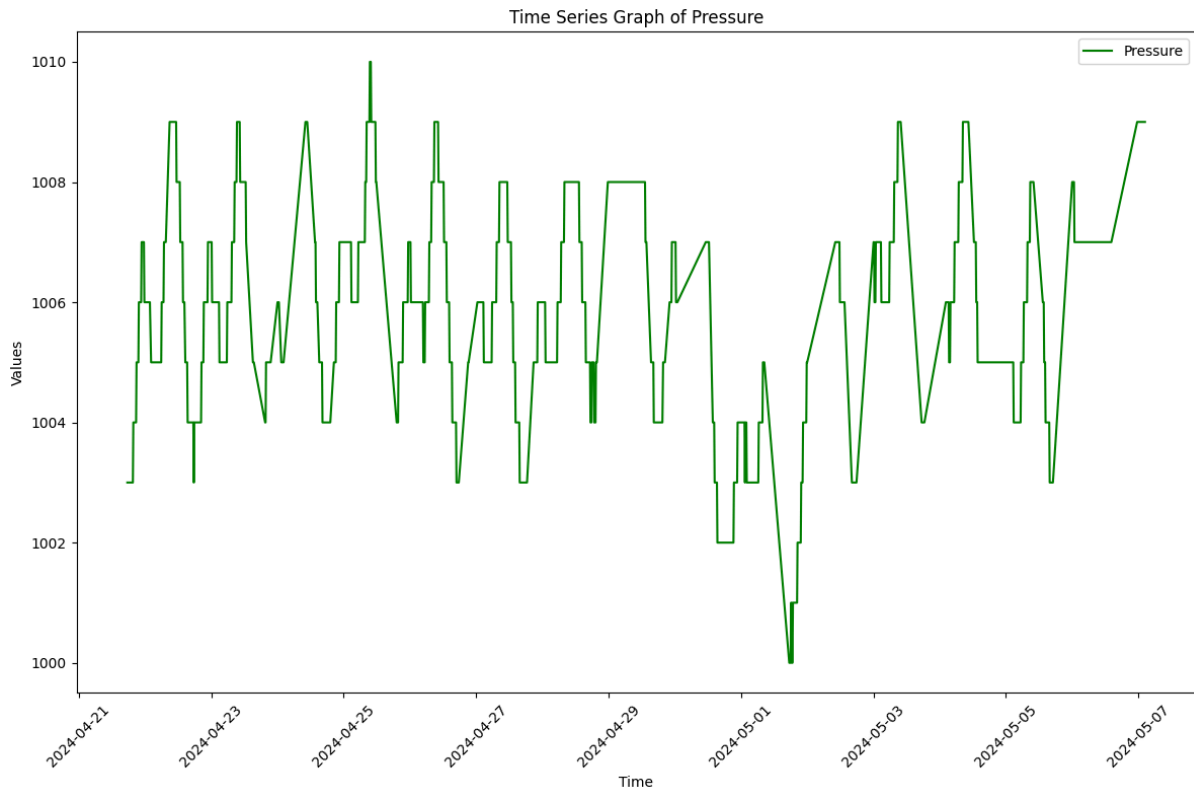


- Use heatmap to find missing data.

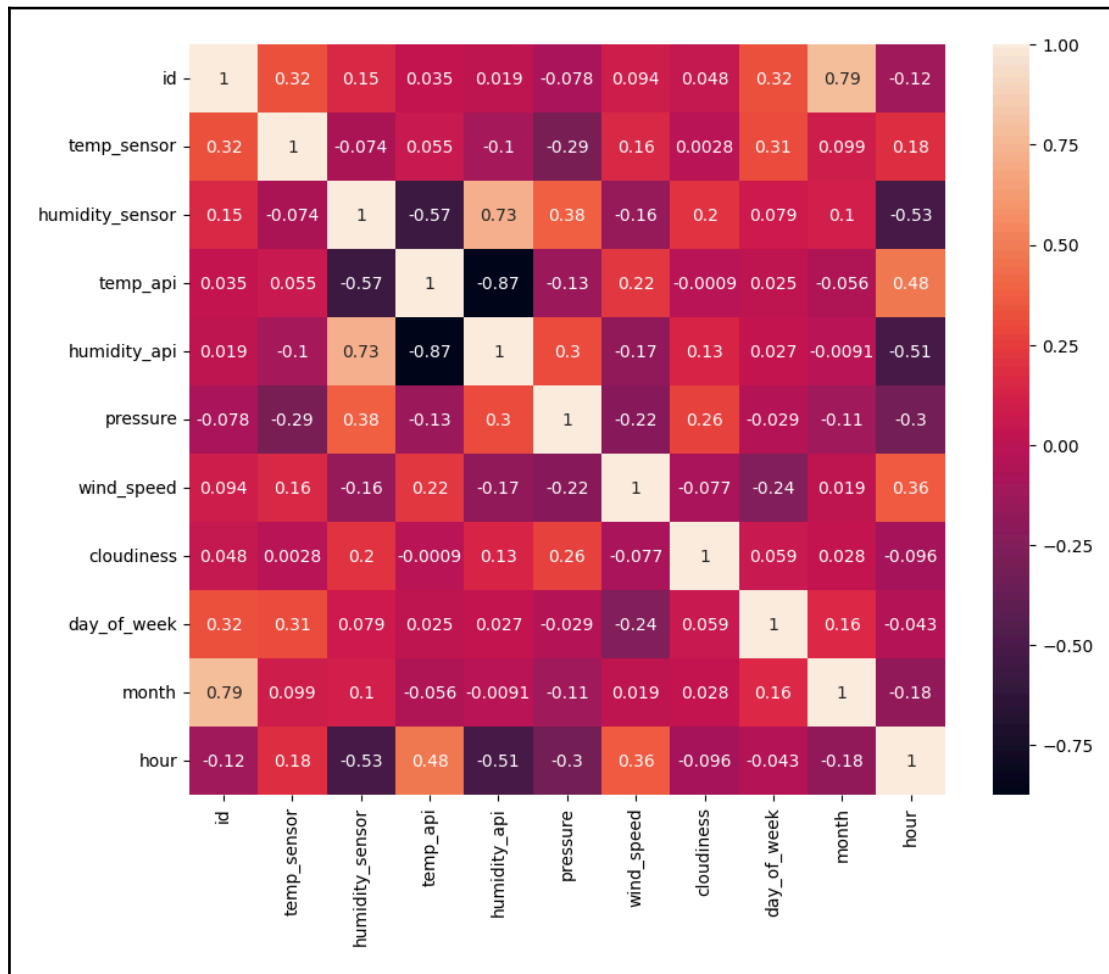


- Plot time series graph among numerical features to find a trend of data in the future.





- Use Heatmap to show correlation among numerical features



- Find average percentage error of temperature from sensor and api

7.75 Percent

- Find average percentage error of humidity from sensor and api

19.21 Percent


Data Preprocessing

- Convert ts to new columns day_of_week, month, hour then drop ts
because we need use these data columns for training model

A terminal window with a dark background and three colored window control buttons (red, yellow, green) at the top left. It contains six lines of Python code for data manipulation.

```
1 data["day_of_week"] = data["ts"].dt.dayofweek
2 data["month"] = data["ts"].dt.month
3 data["hour"] = data["ts"].dt.hour
4
5 data.drop("ts", axis=1, inplace=True)
6 display(data.head())
```

- No need to handle outliers due to in dataset different weather can make cause of extreme value examples.
 - The humidity on a sunny day must be low but on a rainy day it is very high etc.
- Predictor and target split:

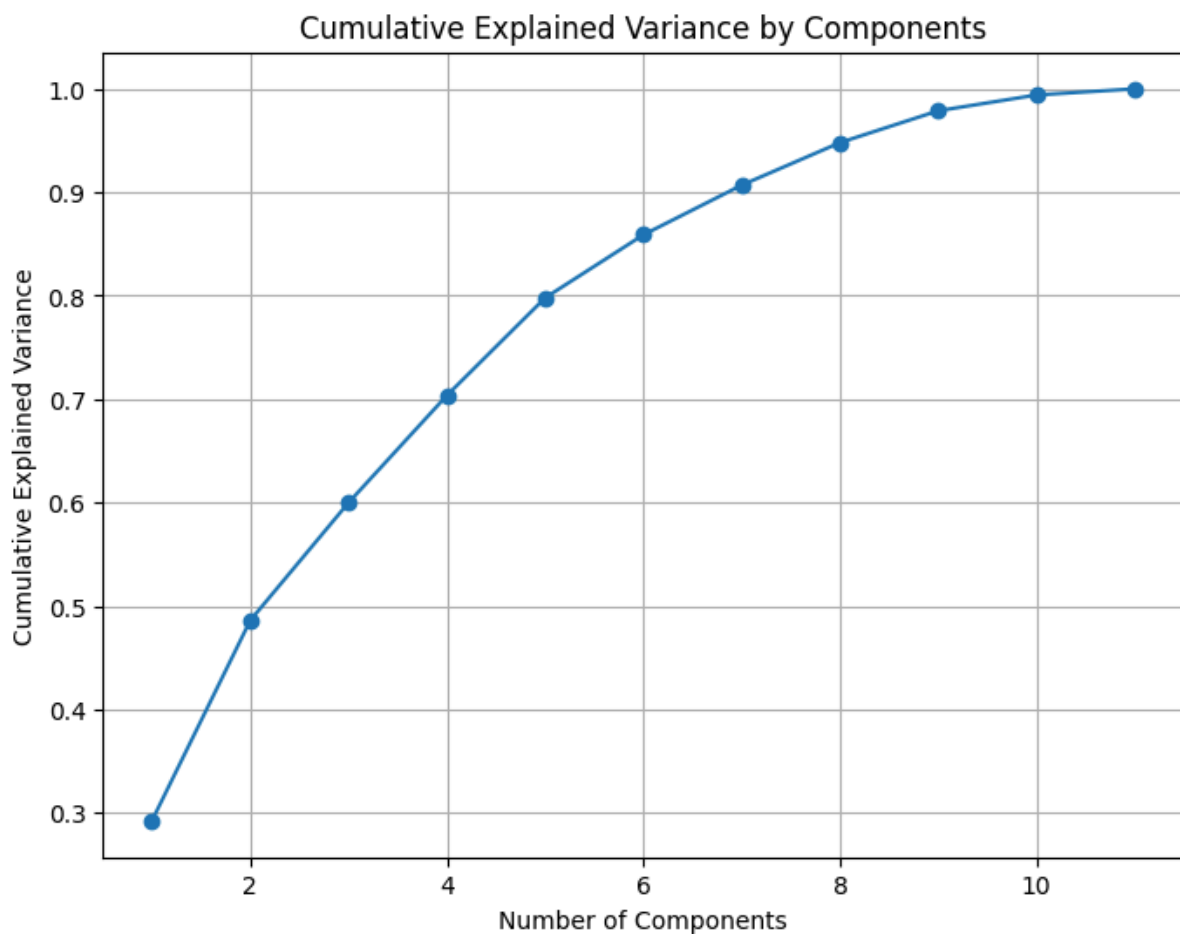
A terminal window with a dark background and three colored window control buttons (red, yellow, green) at the top left. It contains two lines of Python code for splitting the data into predictor and target variables.

```
1 X = data.drop(["weather"], axis=1)
2 y = data["weather"]
```

- Scale data with standard scaler before pca technique step

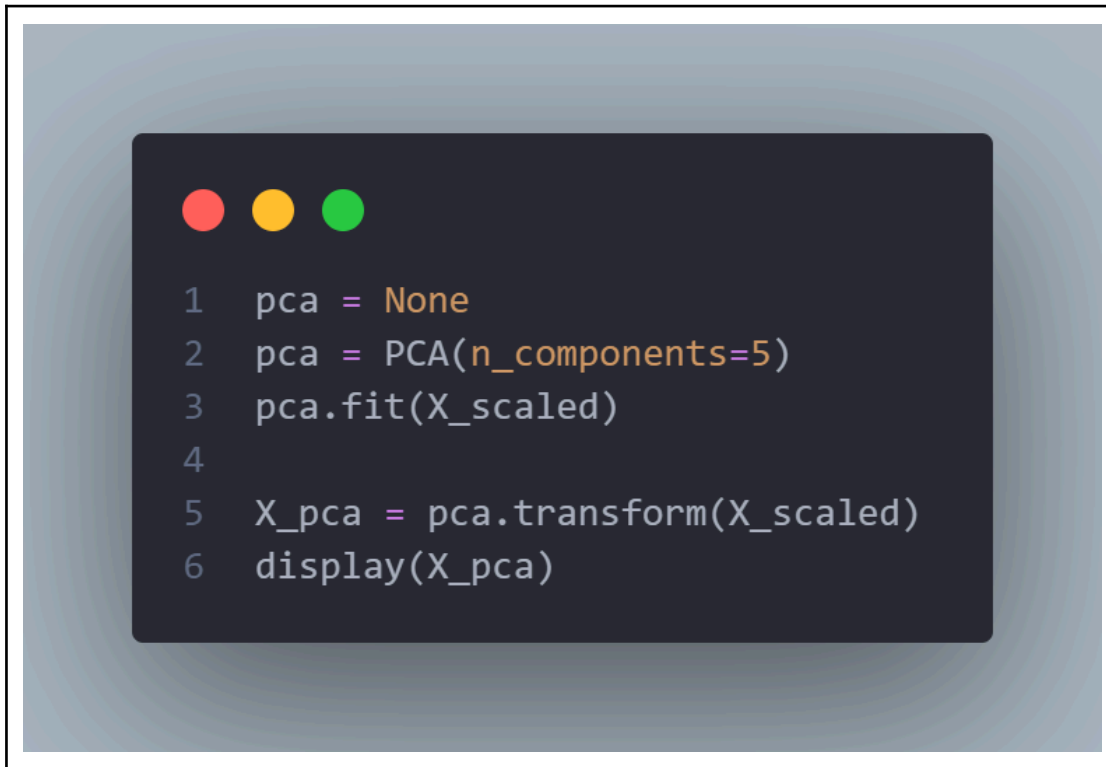
```
1 # Scale data (X)
2 standard_scaler = StandardScaler()
3 X_scaled = standard_scaler.fit_transform(X)
4 display(X_scaled)
```

- Plot cumulative variance by component to select number of component to using in PCA technique



- From plot we select number of component is 5 because it near to 80 percent

- Then We adapt PCA 5 components to our data



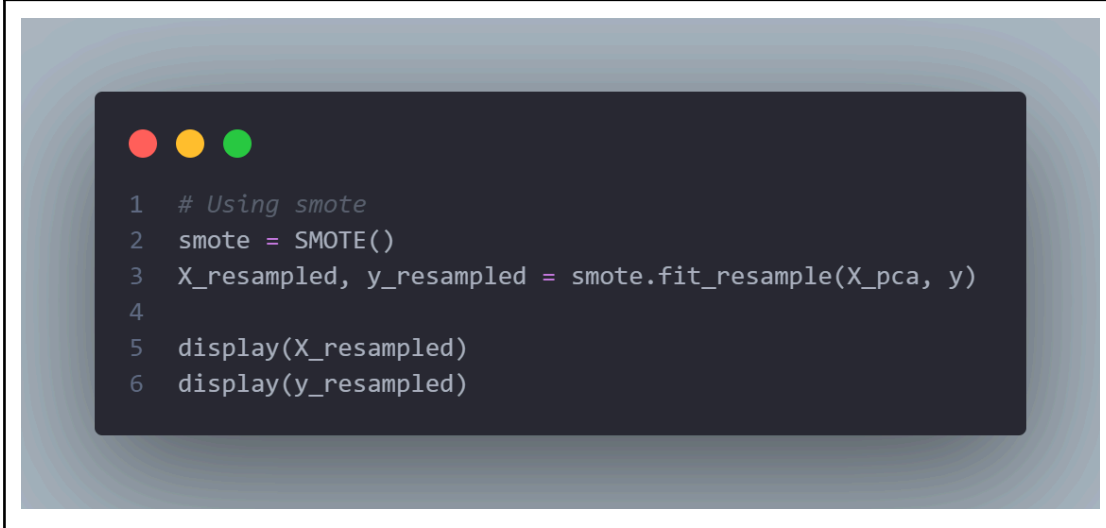
```
1  pca = None
2  pca = PCA(n_components=5)
3  pca.fit(X_scaled)
4
5  X_pca = pca.transform(X_scaled)
6  display(X_pca)
```

- Using a **SMOTE** technique to prevent Imbalanced Data.

Imbalanced data : refers to a situation in classification problems where the distribution of classes in the dataset is highly skewed, meaning that one class is significantly more prevalent than the others. This imbalance can lead to biased models that perform poorly in accurately predicting the minority class, as the model may become overly biased towards the majority class.

SMOTE : (Synthetic Minority Over-sampling Technique) is a method used to address class imbalance in datasets by generating synthetic examples of the minority class. It works by creating new synthetic instances along the line segments joining existing minority class instances, thereby balancing the class distribution and improving the performance of machine learning models on imbalanced datasets.

Why SMOTE?: During summer, our data collection predominantly reflects sunny conditions with fewer instances of rain. This imbalance can arise due to the larger volume of sunny data compared to rainy data.

A code editor window with a dark background and three colored window control buttons (red, yellow, green) at the top left. The code is as follows:

```
1 # Using smote
2 smote = SMOTE()
3 X_resampled, y_resampled = smote.fit_resample(X_pca, y)
4
5 display(X_resampled)
6 display(y_resampled)
```

Modeling

We use Random Forest classification techniques to predict weather because Random Forest is a type of machine learning that creates a group of decision trees. It's straightforward to use and often gives excellent results without needing fine-tuning.

Pros:

- Versatility: Random Forests can do both classification and regression tasks.
- Data Compatibility: Works with categorical and numerical data without needing scaling.
- Feature Selection: Automatically picks relevant features.

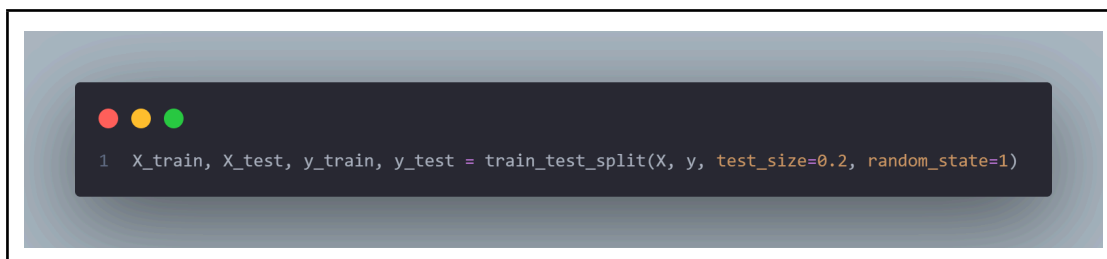
- **Outlier Resilience:** Handles outliers well.
- **Relationship Handling:** Works with linear and non-linear relationships.
- **Accuracy:** Often provides high accuracy.
- **Bias-Variance Balance:** Balances bias and variance effectively.

Cons:

- **Interpretability:** Not easy to interpret like linear regression.
- **Computationally Intensive:** Can be slow for large datasets.
- **Black Box Nature:** Limited control over model workings.

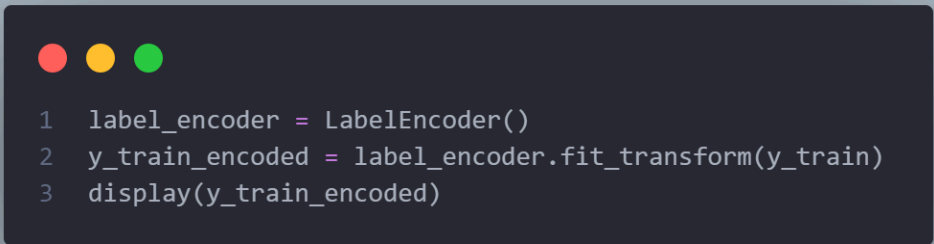
Modeling Step

- **setup before start:**
 - Predictors are all attributes
 - Target “weather”
- **Split train test technique:**



```
1 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)
```

- Use **LabelEncoder** to encode **y_train** (weather) that categorical data to numerical data to Use to train model




```

1  label_encoder = LabelEncoder()
2  y_train_encoded = label_encoder.fit_transform(y_train)
3  display(y_train_encoded)

```

- Next step i choose “Grid search CV technique” to find a best n-estimate of RandomForestClassification model

GridSearchCV helps find the best settings for a model by trying different options and picking the one that works best. It's like testing different ingredients for a recipe to make the tastiest dish.



```

1  param_grid = {
2      'n_estimators': range(10, 201, 10)
3  }
4
5  # Create a Random Forest classifier
6  rf_classifier = RandomForestClassifier(random_state=1)
7
8  # Perform grid search with cross-validation
9  grid_search = GridSearchCV(estimator=rf_classifier, param_grid=param_grid, cv=5, scoring='accuracy')
10 grid_search.fit(X_train, y_train_encoded)
11
12 # Get the best parameters and best score
13 best_n_estimators = grid_search.best_params_['n_estimators']
14 best_score = grid_search.best_score_
15
16 print("Best Number of Estimators:", best_n_estimators)
17 print("Best Accuracy Score:", best_score)

```

- After finding the best n-estimate we training a model again

```
1 rf_classifier = None
2 rf_classifier = RandomForestClassifier(n_estimators=best_n_estimators, random_state=1)
3 rf_classifier.fit(X_train, y_train_encoded)
```

- Prediction X_test

```
1 predictions_encoded = rf_classifier.predict(X_test)
2 predictions = label_encoder.inverse_transform(predictions_encoded)
3 display(predictions)
```

- Evaluate the model

```
1
2 accuracy = accuracy_score(y_test, predictions)
3 print("Accuracy:", accuracy)
4 print("-----")
5 class_report = classification_report(y_test, predictions)
6 print("Classification Report:\n", class_report)
7 print("-----")
8 conf_matrix = confusion_matrix(y_test, predictions)
9 print("Confusion Matrix:\n", conf_matrix)
```

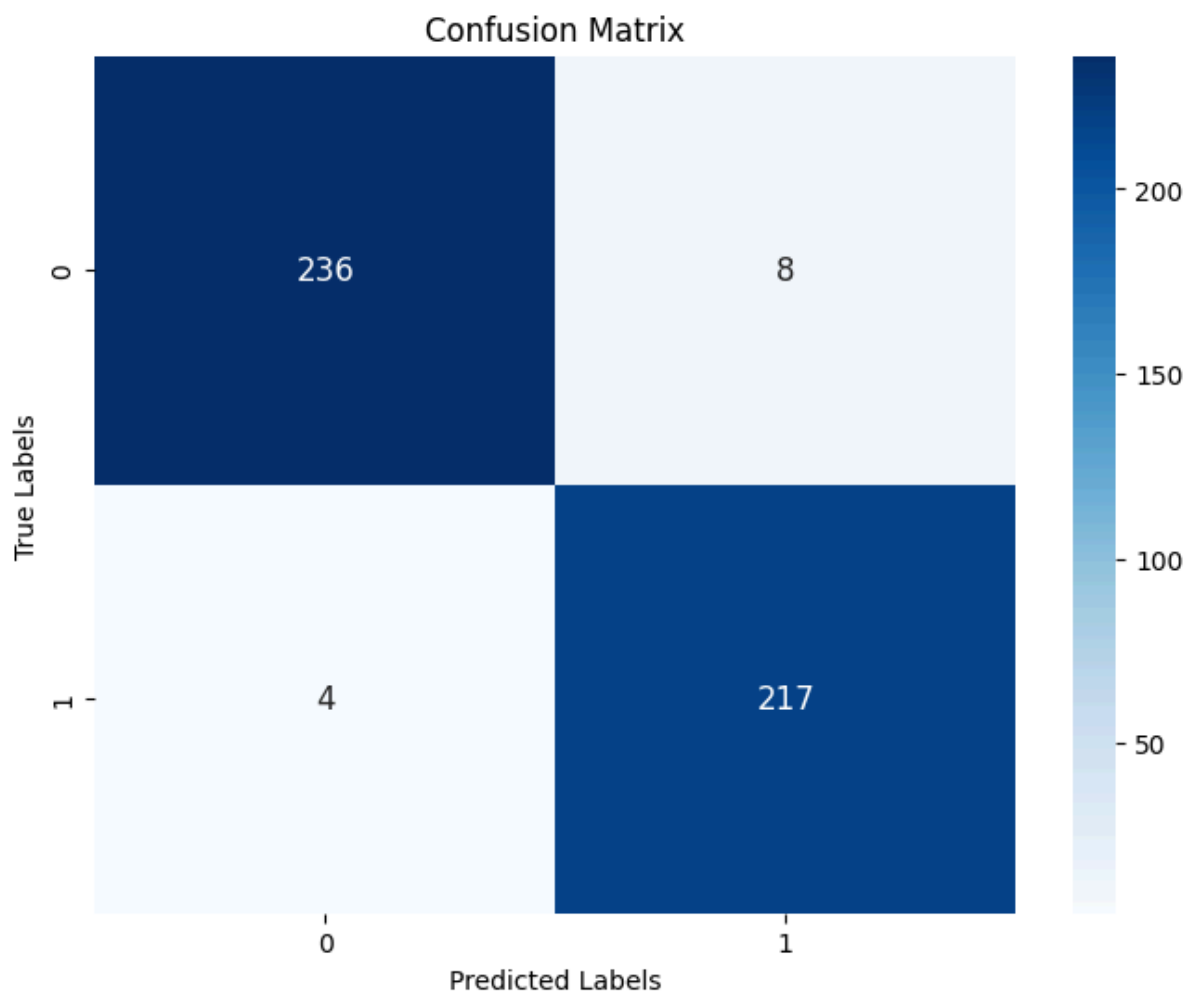
Accuracy: 0.9741935483870968

Classification Report:

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| Clouds | 0.98 | 0.97 | 0.98 | 244 |
| Rain | 0.96 | 0.98 | 0.97 | 221 |
| accuracy | | | 0.97 | 465 |
| macro avg | 0.97 | 0.97 | 0.97 | 465 |
| weighted avg | 0.97 | 0.97 | 0.97 | 465 |

Confusion Matrix:

```
[[236  8]
 [ 4 217]]
```

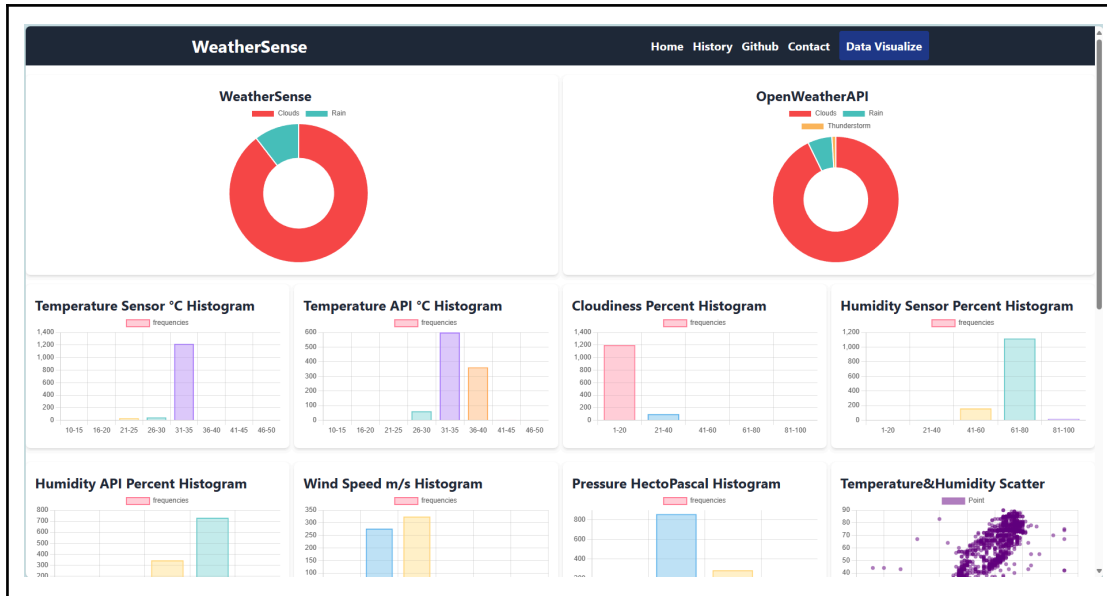


Possible Application

- **Forecasting:** Providing accurate weather forecasts for various locations and time intervals, helping individuals and organizations plan their activities accordingly.
- **Agriculture:** Assisting farmers in making informed decisions about planting, harvesting, irrigation, and pest control based on weather predictions.
- **Travel:** Helping travelers plan their trips by providing weather forecasts for their destinations, ensuring they have a pleasant experience.
- **Transportation:** Enhancing transportation safety and efficiency by predicting weather-related hazards such as storms, heavy rainfall, or snowfall.

WeatherSense website

| WeatherSense | | Home History Github Contact | |
|--------------------------------------------|----------------------------------------|---------------------------------------------------------------------------------------------|------------------------------|
| Latest update on April 20, 2024 at 3:10 AM | | | |
| WeatherSens Clouds | | OpenWeatherMap few clouds | |
| Temperature WeatherSense 28 °C | Temperature OpenWeatherMap 31.74 °C | | Pressure 1005 Hectopascal |
| Humidity WeatherSense 64 Percent | Humidity OpenWeatherMap 89 Percent | | Wind Speed 4.12 m/s |
| Cloudiness 20 Percent | API 30 degree Celsius | | API 30 degree Celsius |



Reference

- <https://medium.datadriveninvestor.com/random-forest-pros-and-cons-c1c42fb64f04>
- [ปรับ Parameters ของโมเดล Machine Learning ด้วย GridSearchCV ใน Scikit-Learn | by Kan Ouivirach | Medium](#)
- [ปัญหาข้อมูลไม่สมดุล \(Imbalanced Data in Classification Model\) - NT Cloud Solutions \(ntplc.co.th\)](#)