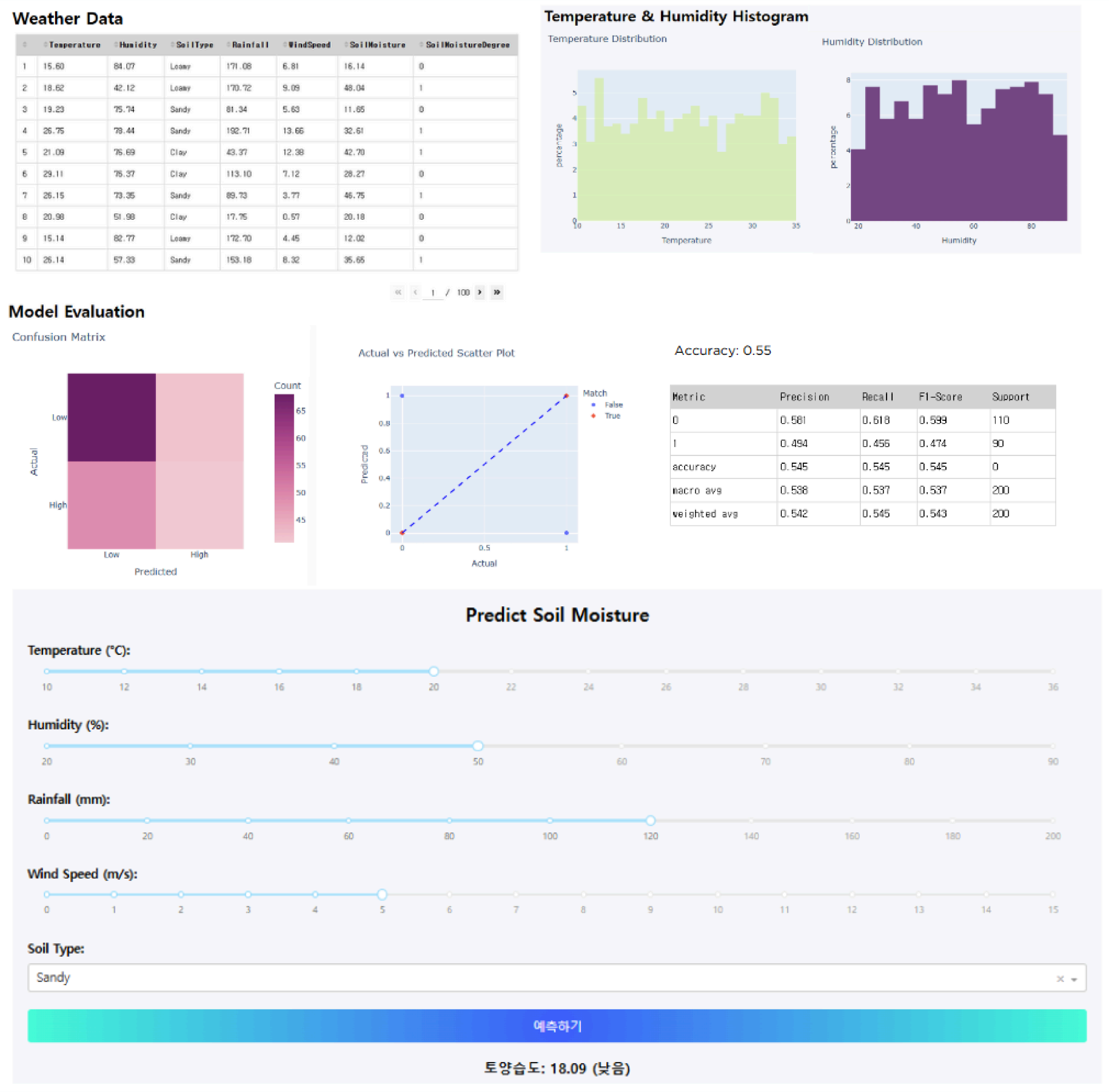
Smart Farm Soil Moisture Prediction Tutorial



Weather Data

	Temperature	Humidity	SoilType	Rainfall	WindSpeed	SoilMoisture	SoilMoistureDegree
1	15.60	84.07	Loamy	171.08	6.81	16.14	0
2	18.62	42.12	Loamy	170.72	9.09	48.04	1
3	19.23	75.74	Sandy	81.34	5.63	11.65	0
4	26.75	78.44	Sandy	192.71	13.66	32.61	1
5	21.09	76.69	Clay	43.37	12.38	42.70	1
6	29.11	76.37	Clay	113.10	7.12	28.27	0
7	26.15	73.35	Sandy	89.73	3.77	46.75	1
8	20.98	51.98	Clay	17.75	0.57	20.18	0
9	15.14	82.77	Loamy	172.70	4.45	12.02	0
10	26.14	57.33	Sandy	153.18	8.32	35.65	1

Temperature & Humidity Histogram

Model Evaluation

Accuracy: 0.55

Metric	Precision	Recall	F1-Score	Support
0	0.581	0.618	0.599	110
1	0.494	0.456	0.474	90
accuracy	0.545	0.545	0.545	0
macro avg	0.538	0.537	0.537	200
weighted avg	0.542	0.545	0.543	200

Predict Soil Moisture

Temperature (°C):

Humidity (%):

Rainfall (mm):

Wind Speed (m/s):

Soil Type:

Sandy

토양습도: 18.09 (낮음)

Objectives

Smart farm can utilize environmental data to monitor and predict soil moisture. Implement a logistic regression model to predict whether soil moisture levels are high or low based on data you will create, Visualize it as a dashboard.

1: Creating and Exploring Datasets

Create a data frame using pandas and numpy using the data given below.
(The number of data is 1000 each.)

- Temperature (range 10~35°C) => Temperature
- Humidity (20~90% range) => Humidity
- Soil type (Sandy, Clay, Loamy) => Soil variable SoilType, type ['Sandy', 'Clay', 'Loamy']
- Precipitation (range 0~200 mm) => Rainfall
- Wind speed (range 0~15 m/s) => WindSpeed
- Soil humidity (range 10~50%) => SoilMoisture

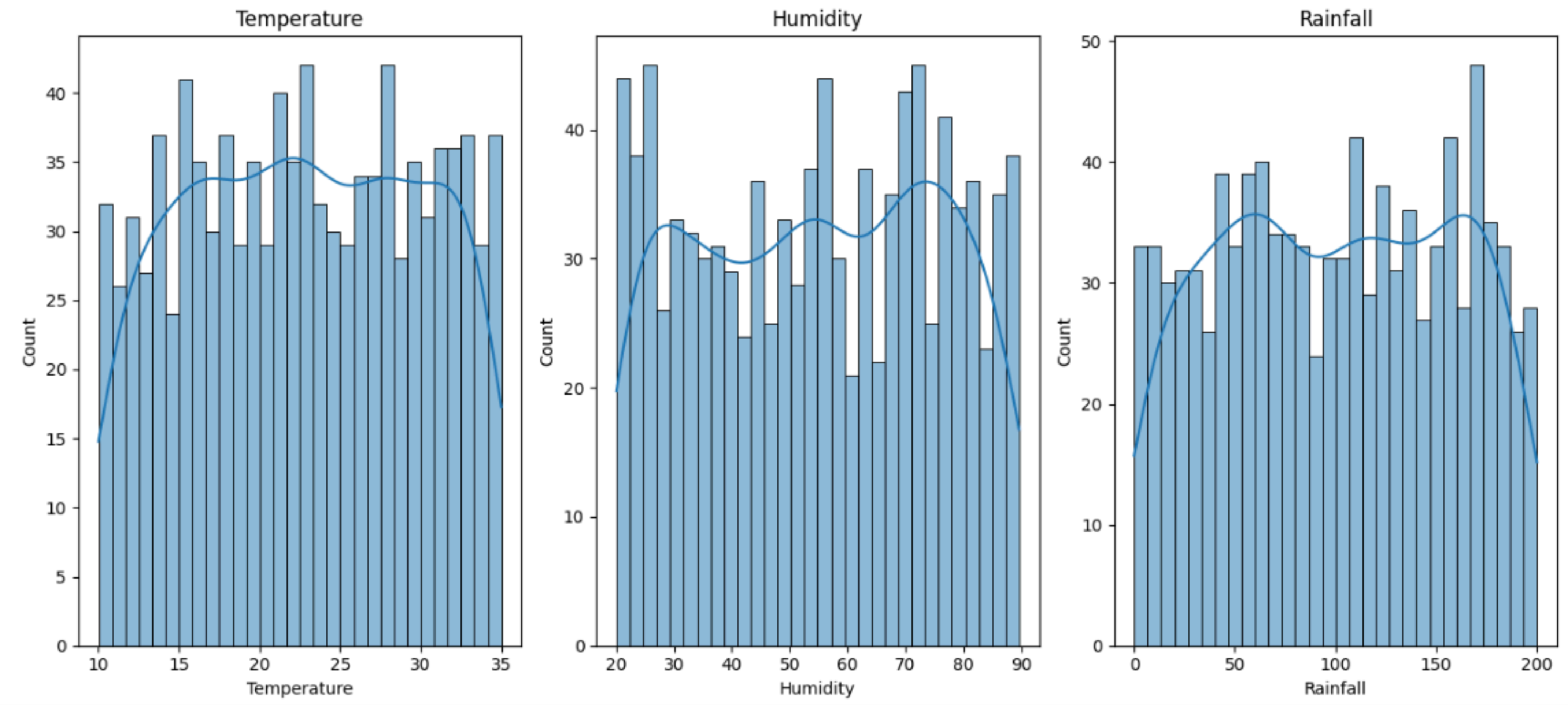
requirements

- Print the first five rows of the generated dataset.

- Check the basic statistics of the data (using '.describe()') and check for missing values.
- Visualize the distribution of 'Temperature', 'Humidity', and 'Rainfall' as a histogram.

```
# collab
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# 1. 데이터 생성
MAX_SIZE = 1000
soil_types = ['Sandy', 'Clay', 'Loamy']

weather_dict = {
    'Temperature': np.random.uniform(10, 35, MAX_SIZE).tolist(),
    'Humidity': np.random.uniform(20, 90, MAX_SIZE).tolist(),
    'SoilType': np.random.choice(soil_types, size=MAX_SIZE).tolist(),
    'Rainfall': np.random.uniform(0, 200, MAX_SIZE).tolist(),
    'WindSpeed': np.random.uniform(0, 15, MAX_SIZE).tolist(),
    'SoilMoisture': np.random.uniform(10, 50, MAX_SIZE).tolist()
}

df = pd.DataFrame(weather_dict)

# 2. 데이터셋의 처음 5개 행 출력
print("처음 5개 행:\n", df.head())

# 3. 데이터 기본 통계 확인 및 결측값 점검
print("\n기본 통계:\n", df.describe())
print("\n결측값 여부:\n", df.isnull().sum())

# 4. 'Temperature', 'Humidity', 'Rainfall'의 분포를 히스토그램으로 시각화
plt.figure(figsize=(13, 6))
for i, col in enumerate(['Temperature', 'Humidity', 'Rainfall']):
    plt.subplot(1, 3, i + 1)
    sns.histplot(df[col], kde=True, bins=30, alpha=0.5)
    plt.title(f'{col}')
plt.tight_layout()
plt.show()
```

```
처음 5개 행:
    Temperature   Humidity SoilType    Rainfall  WindSpeed  SoilMoisture
0    26.896728  75.819363    Sandy   82.055911  11.993707     15.994439
1    25.636575  87.113933     Clay   82.771837   0.933470     16.175567
2    23.466401  26.530198     Clay  108.657674  11.153672     25.178655
3    33.327757  80.533965    Loamy   10.011543   0.163955     34.152241
4    19.684246  39.535942    Loamy  125.081637   0.876312     37.257171

기본 통계:
        Temperature     Humidity     Rainfall    WindSpeed  SoilMoisture
count  1000.000000  1000.000000  1000.000000  1000.000000   1000.000000
mean     22.749019    54.811090   100.545332     7.460358     30.530895
std       7.119887    20.601651    57.045995     4.278176     11.781820
min      10.007779    20.042533     0.045427     0.002412     10.056851
25%      16.638168    36.428490    52.048915     3.752343     20.114446
50%      22.760063    55.264473   101.474081     7.467382     30.617053
75%      28.834184    72.671865   152.393464    11.183821     40.865565
max      34.987810    89.684556   199.989511    14.979476     49.982608

결측값 여부:
 Temperature     0
Humidity        0
SoilType        0
Rainfall        0
WindSpeed       0
SoilMoisture    0
dtype: int64
```



2: Preprocessing Datasets

dev env

- Flask
- Pycharm community

app.py

```
import dash
from model import train_model

# server
server = Flask(__name__)

# create and train model
df = train_model()

# run server
if __name__ == '__main__':
    server.run(debug=True)
```

config.py

```
import numpy as np

MAX_SIZE = 1000
SOIL_TYPES = ['Sandy', 'Clay', 'Loamy']
TARGET_THRESHOLD = 30  # SoilMoisture 변수를 30% 기준
weather_dict = {
    'Temperature': np.random.uniform(10, 35, MAX_SIZE).tolist(),
    'Humidity': np.random.uniform(20, 90, MAX_SIZE).tolist(),
    'SoilType': np.random.choice(SOIL_TYPES, size=MAX_SIZE).tolist(),
    'Rainfall': np.random.uniform(0, 200, MAX_SIZE).tolist(),
    'WindSpeed': np.random.uniform(0, 15, MAX_SIZE).tolist(),
    'SoilMoisture': np.random.uniform(10, 50, MAX_SIZE).tolist()
}
```

import libraries and set global variables in model.py

```
import config
import pandas as pd
import numpy as np

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
from sklearn.neighbors import NearestNeighbors

from dash import dash, callback, Output, Input

import plotly.express as px

# global variables
df = None
model = None
X_train = None
X_test = None
y_train = None
y_test = None
```

requirements

- Convert a categorical variable, such as SoilType, to a number by one-hot encoding it.
- Binary classify the SoilMoisture variable based on 30% (0: low, 1: high).
- Split the data into a training set (80%) and a test set (20%).
- Perform data normalization and print the mean and standard deviation.

train_model() in model.py

```
def train_model():
    global df, model, X_train, X_test, y_train, y_test
    df = pd.DataFrame(config.weather_dict)
    # print(df.head())
    # print(df.describe())

    # SoilMoisture 이진 분류 생성
    df['SoilMoistureDegree'] = np.where(df['SoilMoisture'] > config.TARGET_THRESHOLD, 1, 0)

    # 특성과 타겟 분리
    X = df.drop(columns=['SoilMoisture', 'SoilMoistureDegree'])
    y = df['SoilMoistureDegree']

    # 데이터 분할
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # 토양 유형(SoilType) 원-핫 인코딩 및 표준화 스케일링을 포함하는 파이프라인 정의
    # OneHotEncoder를 사용하여 훈련 세트와 테스트 세트에 대해 동일한 인코딩을 적용
    preprocessor = ColumnTransformer(
        transformers=[
            ('num', StandardScaler(), ['Temperature', 'Humidity', 'Rainfall', 'WindSpeed']),
            ('cat', OneHotEncoder(drop='first'), ['SoilType'])
        ]
    )

    # 모델 파이프라인 정의 (로지스틱회귀)
    model = Pipeline(steps=[
        ('preprocessor', preprocessor),
        ('classifier', LogisticRegression(max_iter=1000, random_state=42))
    ])

    # 모델 학습
    model.fit(X_train, y_train)

    # 학습된 모델 출력
    # 1. 전처리기(Preprocessor) 가져오기
    preprocessor = model.named_steps['preprocessor']

    # 2. 수치형 변수의 평균과 표준편차 출력
    if hasattr(preprocessor.named_transformers_['num'], 'mean_'):
        print("수치형 변수 평균:", preprocessor.named_transformers_['num'].mean_)
        print("수치형 변수 표준편차:", preprocessor.named_transformers_['num'].scale_)

    # 3. 원-핫 인코더의 변환된 클래스 출력
    if hasattr(preprocessor.named_transformers_['cat'], 'categories_'):
        print("원-핫 인코딩된 클래스:", preprocessor.named_transformers_['cat'].categories_)

    .
    .
    .
    return df
```

```
수치형 변수 평균: [ 22.64529429  55.32044726 101.57732775   7.47171436]
수치형 변수 표준편차: [ 7.18744874 20.95467849 57.21062832  4.20746486]
원-핫 인코딩된 클래스: [array(['Clay', 'Loamy', 'Sandy'], dtype=object)]
```

3. Implement Logistic Regression Model

requirements

- Train the model using LogisticRegression. (Random seed: 42, maximum number of iterations: 1000)
- Make predictions using test data.
- Print the model’s accuracy.

train_model() in model.py

```
def train_model():
    .
    .
    .
    # 모델 파이프라인 정의 (로지스틱회귀)
    model = Pipeline(steps=[
        ('preprocessor', preprocessor),
        ('classifier', LogisticRegression(max_iter=1000, random_state=42))
    ])

    # 모델 학습
    model.fit(X_train, y_train)

    # 학습된 모델 출력
    .
    .
    .
    # 4. 모델의 계수 및 정확도 출력 (Logistic Regression)
    model_coefficients = model.named_steps['classifier'].coef_
    model_intercept = model.named_steps['classifier'].intercept_

    print("모델 계수:", model_coefficients)
    print("모델 절편:", model_intercept)

    accuracy = accuracy_score(y_test, model.predict(X_test))
    print("모델 정확도:", accuracy)

    return df
```

```
모델 계수: [[-0.01166415 -0.02955629 -0.01722798 -0.07567536 -0.2606666  -0.03820069]]
모델 절편: [0.02364861]
모델 정확도: 0.485  # avg 0.45 to 0.55
```

4.Model evaluation and performance analysis

requirements

- Create and visualize a confusion matrix.
- Calculate precision, recall, and F1-score.
- Create a scatterplot comparing actual and predicted values.

show_model_evaluation_and_confusion_matrix(n_clicks) in model.py

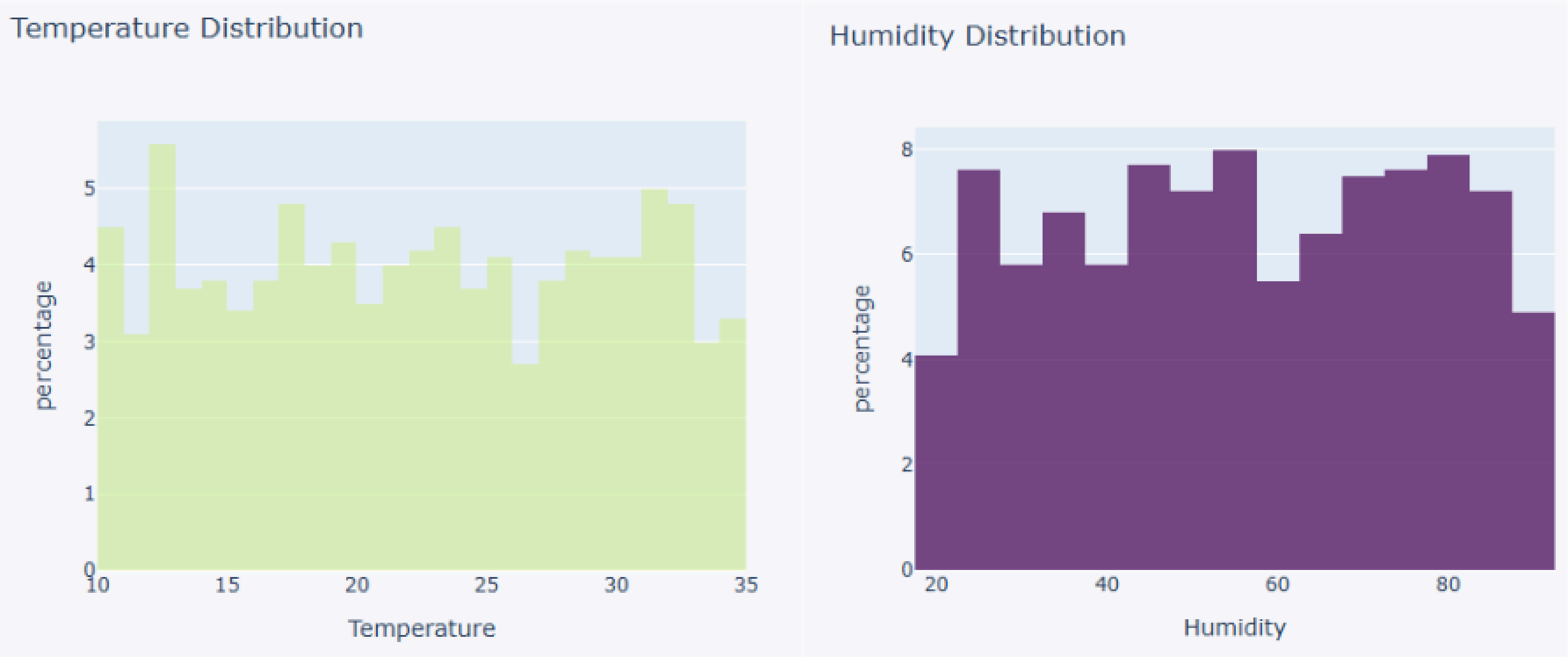
```
@callback(
    Output('model-evaluation', 'children'),
    Output('confusion-matrix', 'figure'),
    Output('classification-report-table', 'data'),
    Output('scatter-plot', 'figure'),
    Input('predict-button', 'n_clicks')
)
def show_model_evaluation_and_confusion_matrix(n_clicks):
    global model, X_train, X_test, y_train, y_test

    if n_clicks:
        return dash.no_update, dash.no_update, dash.no_update, dash.no_update

    print("model-evaluation is called...")

    # 정확도 계산
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    conf_matrix = confusion_matrix(y_test, y_pred)

    print("model: ", model)
    print("accuracy: ", accuracy)
    print("conf_matrix: ", conf_matrix)

    # classification_report 출력
    report = classification_report(y_test, y_pred, output_dict=True)
    print("Classification Report:")
    print(report)

    # classification_report 포맷팅 함수 호출
    report_data = format_classification_report(report)

    # 혼동 행렬 시각화
    fig = px.imshow(conf_matrix,
                    labels=dict(x="Predicted", y="Actual", color="Count"),
                    x=['Low', 'High'],
                    y=['Low', 'High'],
                    color_continuous_scale=px.colors.sequential.Magenta,
                    title='Confusion Matrix')

    # 실제와 예측된 값을 비교하는 산점도 생성
    scatter_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
    # fig_scatter = px.scatter(scatter_df, x='Actual', y='Predicted',
    #                          labels={'Actual': '실제 값', 'Predicted': '예측 값'},
    #                          title='실제 vs 예측 값 산점도')

    # 일치 여부 컬럼 추가
    scatter_df['Match'] = scatter_df['Actual'] == scatter_df['Predicted']

    # 산점도 생성
    fig_scatter = px.scatter(scatter_df, x='Actual', y='Predicted',
                             color='Match',
                             color_continuous_scale=['red', 'green'],
                             labels={'Actual': 'Actual', 'Predicted': 'Predicted'},
                             title='Actual vs Predicted Scatter Plot',
                             symbol='Match',
                             size_max=10)

    # 산점도 대각선 추가
    max_val = max(scatter_df['Actual'].max(), scatter_df['Predicted'].max())
    fig_scatter.add_shape(type='line',
                          x0=0, y0=0, x1=max_val, y1=max_val,
                          line=dict(color='blue', width=2, dash='dash'))

    return f"Accuracy: {accuracy:.2f}", fig, report_data, fig_scatter
```

5: Implementing a Dashboard

Build a dashboard with the following features using Dash:

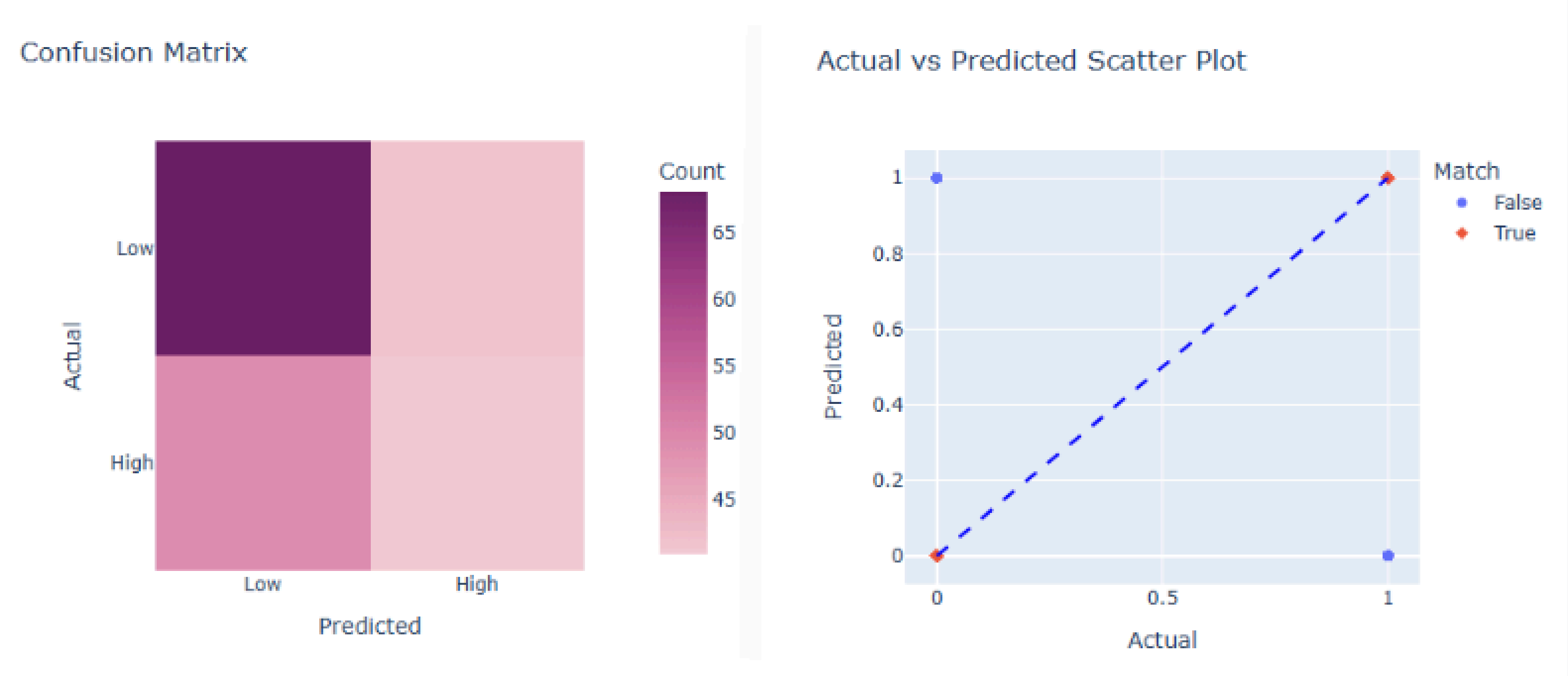
a. Data Section

- Displays distribution of temperature and humidity as a histogram.



b. Model Evaluation Section

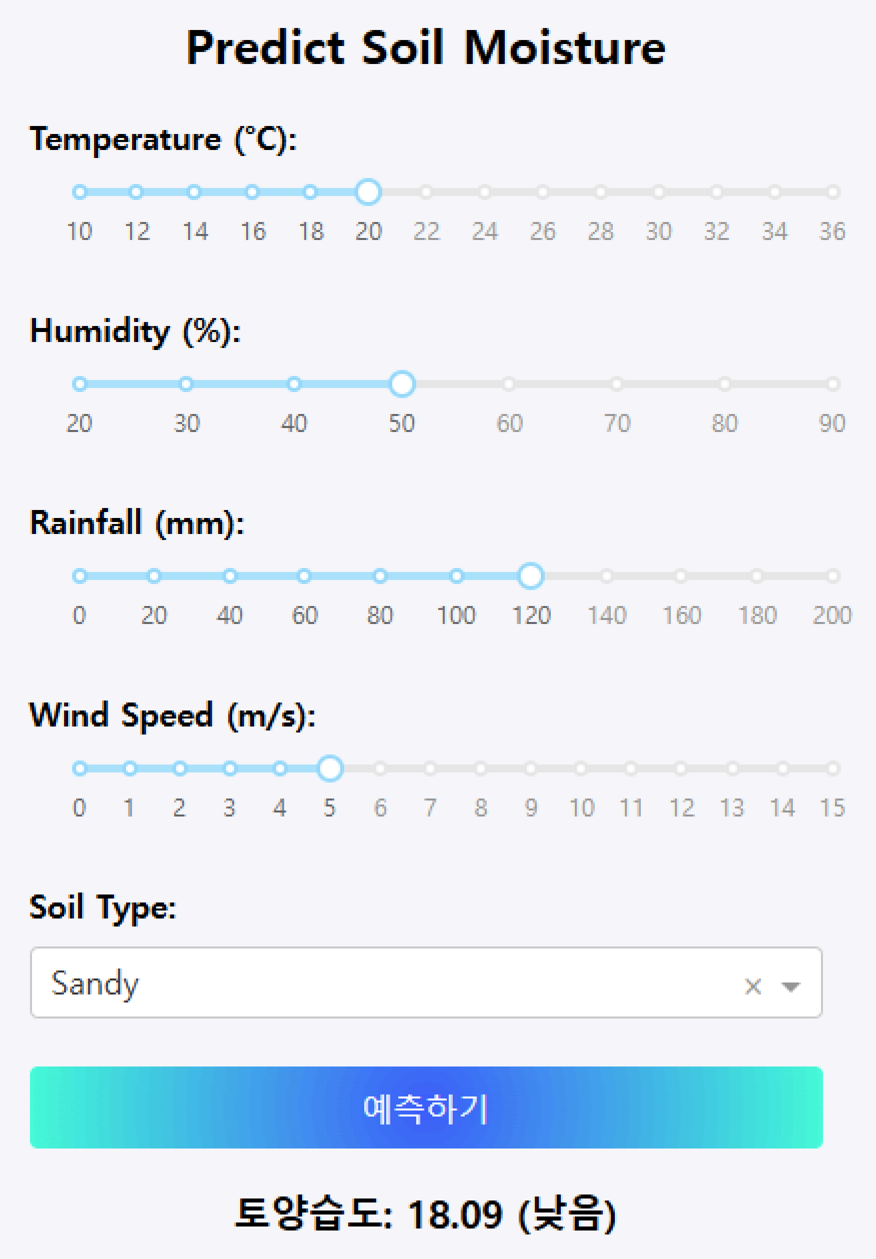
- accuracy and confusion matrix.



Metric	Precision	Recall	F1-Score	Support
0	0.581	0.618	0.599	110
1	0.494	0.456	0.474	90
accuracy	0.545	0.545	0.545	0
macro avg	0.538	0.537	0.537	200
weighted avg	0.542	0.545	0.543	200

c. Prediction Section based on User Input

- Outputs prediction results on user inputs temperature, humidity, precipitation, wind speed, and soil type.
- Test predictions based on user input values using `predict()`



Predict Soil Moisture

Temperature (°C):

Humidity (%):

Rainfall (mm):

Wind Speed (m/s):

Soil Type:

Sandy

토양습도: 18.09 (낮음)

layout.py

```
import config
from dash import dcc, html
from dash.dash_table import DataTable
import plotly.express as px


def create_histogram_graph(_id, df, x_col, title):
    # 히스토그램 그래프 생성 함수

    print(df.head())
    fig = px.histogram(
        df,
        x=x_col,
        nbins=30,
        title=title,
        color_discrete_sequence=px.colors.sequential.Emrld if x_col == 'Temperature' else
px.colors.sequential.Viridis,
        opacity=0.7,
        histnorm='percent',
    )
    fig.update_layout(
        # plot_bgcolor='lightblue',
        paper_bgcolor='rgb(248, 249, 250)',
        xaxis_title=x_col,
        yaxis_title='percentage'
    )
    return dcc.Graph(
        id=_id,
        figure=fig
    )


def create_slider(_id, label, min_value, max_value, value, marks):
    # 슬라이더 생성 함수
    return html.Div([
        html.Label(label, className='label'),
        dcc.Slider(
            id=_id,
            min=min_value,
            max=max_value,
            value=value,
            marks=marks,
            step=1,
            className='input'
        )
    ])


def create_dropdown(_id, label, options, value):
    # 드롭다운 생성 함수
    return html.Div([
        html.Label(label, className='label'),
        dcc.Dropdown(
            id=_id,
            options=options,
            value=value,
            className='input'
        )
    ])


def show_weather_data_table(df):
    # 날씨 데이터 테이블로 보여주는 함수

    weather_df = df.copy()
    # 소수점 2자리로 포맷팅
    for col in weather_df.select_dtypes(include=['float64', 'float32']):  # float 타입 열에 대해
        weather_df[col] = weather_df[col].map(lambda x: f"{x:.2f}")  # 소수점 2자리로 변환

    return DataTable(
        id='weather-table',
        columns=[
            {"name": "", "id": "row_number"},
            *[{"name": i, "id": i} for i in weather_df.columns]
        ],
        data=[
            {**row, "row_number": index + 1}
            for index, row in weather_df.iterrows()
        ],
        page_size=10,
        style_table={'overflowX': 'auto', 'margin': '20px'},
        style_header={
            'backgroundColor': 'lightgrey',
            'fontWeight': 'bold',
            'textAlign': 'center',
            'padding': '10px'
        },
        style_cell={
            'padding': '10px',
            'textAlign': 'left',
        },
        sort_action='native',
        # filter_action='native',
        # row_selectable='multi',
        # selected_rows=[],
    )


def show_classification_report_table():
    # Classification report 테이블 생성 함수
    return html.Div([
        # html.H4("Classification Report"),
        DataTable(
            id='classification-report-table',
            columns=[
                {"name": "Metric", "id": "index"},
                {"name": "Precision", "id": "precision"},
                {"name": "Recall", "id": "recall"},
                {"name": "F1-Score", "id": "f1-score"},
                {"name": "Support", "id": "support"}
            ],
            data=[],  # 초기 데이터는 빈 목록
            style_table={'overflowX': 'auto'},
            style_cell={'textAlign': 'left'},
            style_header={'backgroundColor': 'lightgrey'},
            style_data={'whiteSpace': 'normal', 'height': 'auto'},
            page_size=10  # 페이지 크기 설정 (선택 사항)
        )
    ], title='classification Report')


def create_layout(df):
    # 레이아웃 정의
    return html.Div([

        html.Div([
            html.H1("스마트 농업 토양 습도 예측 대시보드"),

            html.Div([
                html.H2("Weather Data"),
                show_weather_data_table(df),
            ], style={'padding': '5% 5% 0 0'}),

            html.Div([
                html.H2("Temperature & Humidity Histogram"),
                create_histogram_graph('histogram-temp', df, 'Temperature', 'Temperature Distribution'),
                create_histogram_graph('histogram-hum', df, 'Humidity', 'Humidity Distribution'),
            ], style={'backgroundColor': '#f8f9fa', 'paddingTop': '5%'}),

            html.Div([
                html.H2("Model Evaluation"),
                html.H4(id='model-evaluation'),
                dcc.Graph(id='confusion-matrix'),
                dcc.Graph(id='scatter-plot'),
                show_classification_report_table(),
            ], style={'paddingTop': '5%', 'marginBottom': '2%'}),
        ], style={'textAlign': 'center'}),

        html.Div([
            html.H2("Predict Soil Moisture", style={'textAlign': 'center'}),
            html.Div([
                html.Div([
                    create_slider('temp-input', "Temperature (°C):", 10, 36, 20, {i: str(i) for i in range(10, 37, 2)}),
                    create_slider('hum-input', "Humidity (%):", 20, 90, 50, {i: str(i) for i in range(20, 91, 10)}),
                    create_slider('rain-input', "Rainfall (mm):", 0, 200, 120, {i: str(i) for i in range(0, 201, 20)}),
                    create_slider('wind-input', "Wind Speed (m/s):", 0, 15, 5, {i: str(i) for i in range(0, 17)}),
                ], style={'paddingRight': '5%'}),
                create_dropdown('soil-type-input', "Soil Type:",
                                [{'label': soil, 'value': soil} for soil in config.SOIL_TYPES], 'Sandy'),
                html.Button('예측하기', id='predict-button', className='button', style={
                    'background': 'radial-gradient(circle, rgba(63,94,251,1) 0%, rgba(70,252,216,1) 100%)'}),

                html.Div(html.H3(id='prediction-output', className='output', style={'padding-bottom': '5%'}))
            ], style={'width': '80%', 'margin': '0 auto'}),

        ], style={'backgroundColor': '#f8f9fa', 'paddingTop': '5%'}
        ),
    ], style={'marginTop': '50px'})
```