Model RF dan LR

```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedKFold, GridSearchCV

from sklearn.preprocessing import MinMaxScaler,  StandardScaler
from sklearn.feature_selection import SelectKBest, SelectPercentile, mutual_info_classif

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier

from sklearn.pipeline import Pipeline
from sklearn.base import clone
import time

from sklearn.metrics import (
   confusion_matrix,
   ConfusionMatrixDisplay,
   classification_report
)

df_cuaca = pd.read_csv('dpc.csv',header=0)

df_cuaca.head()
print("Jumlah baris, kolom:", df_cuaca.shape)
print("\nTipe data:")
print(df_cuaca.dtypes)
```

```
df_cuaca = df_cuaca.drop(columns=['Tekanan (hPa)', 'pH Tanah', 'Kelembapan Tanah (%)'], errors='ignore')


# Tampilkan 5 baris pertama untuk verifikasi

df_cuaca.head()

# (1) Cek nilai NULL

print("\nJumlah nilai kosong (null) per kolom:\n", df_cuaca.isnull().sum())


# (2) Cek nilai NaN

nan_counts = df_cuaca.isna().sum()

print("\nJumlah nilai NaN per kolom:\n", nan_counts)


median_suhu = df_cuaca['Suhu (°C)'].median()

median_awan = df_cuaca['Awan (%)'].median()


# Isi nilai NaN dengan median

df_cuaca['Suhu (°C)'] = df_cuaca['Suhu (°C)'].fillna(median_suhu)

df_cuaca['Awan (%)'] = df_cuaca['Awan (%)'].fillna(median_awan)

df_cuaca['Kelembapan (%)'] = df_cuaca['Kelembapan (%)'].fillna(median_awan)

df_cuaca['Curah Hujan (mm)'] = df_cuaca['Curah Hujan (mm)'].fillna(median_awan)


# Tampilkan hasil median

print("Median Suhu (°C):", median_suhu)

print("Median Awan (%):", median_awan)


# Cek kembali apakah masih ada NaN

print("\nJumlah nilai kosong setelah penanganan:")

print(df_cuaca.isnull().sum())

# Validasi ulang

print("\nSetelah imputasi, nilai kosong per kolom:\n", df_cuaca.isnull().sum())

before = df_cuaca.shape
```

```
dupes = df_cuaca[df_cuaca.duplicated(keep=False)]
print(f"Jumlah baris duplikat (terhitung ganda): {dupes.shape[0]}")
df_cuaca2 = df_cuaca.drop_duplicates(keep='first')
print("Bentuk data sebelum/sesudah hapus duplikat:", before, "->", df_cuaca.shape)



# Deteksi OUTLIER dengan metode IQR
df_cuaca3 = df_cuaca2.select_dtypes(include=[np.number])
Q1 = df_cuaca3.quantile(0.25)
Q3 = df_cuaca3.quantile(0.75)
IQR = Q3 - Q1
outliers = ((df_cuaca3 < (Q1 - 1.5 * IQR)) | (df_cuaca3 > (Q3 + 1.5 * IQR))).any(axis=1)
print("\nJumlah baris yang terdeteksi memiliki outlier:", outliers.sum())

# Hapus baris yang memiliki outlier
df_cuaca_clean = df_cuaca2.loc[~outliers]
print("Ukuran Data Asli:", df_cuaca2.shape)
print("Ukuran Data Setelah Outlier Dihapus:", df_cuaca_clean.shape)
print("Jumlah baris yang dihapus:", outliers.sum())
# ==============================
# Encode Label & Pisahkan Fitur–Target
# ==============================

# Hujan  → 0 (Cuaca Hujan)
# Cerah  → 1 (Cuaca Cerah)

# Mengubah label prakiraan cuaca dari huruf menjadi angka
df_cuaca2['Prakiraan Cuaca'] = df_cuaca2['Prakiraan Cuaca'].map({'Hujan': 0, 'Cerah': 1})

# Menentukan X sebagai fitur (semua kolom kecuali 'Prakiraan Cuaca')
X = df_cuaca2.drop(columns=['Prakiraan Cuaca'])
```

```
# Menentukan y sebagai target (kolom 'Prakiraan Cuaca')
y = df_cuaca2['Prakiraan Cuaca']

# Menampilkan 5 baris pertama dataframe setelah perubahan
df_cuaca2.head()

X_train, X_test, y_train, y_test = train_test_split(
   X, y, test_size=0.3, random_state = 45, stratify=y
)
print("Ukuran X_train, X_test:", X_train.shape, X_test.shape)
# ============================================================
#  PIPELINE: Scaling → Feature Selection → Logistic Regression
# ============================================================

# Rancang pipeline: gabungkan preprocessing + feature selection + model
pipe_lr = Pipeline(steps=[
   ('scaler', MinMaxScaler()),
   ('feat_select', SelectKBest()),
   ('clf', LogisticRegression(     # model klasifikasi
      class_weight='balanced',
      solver='liblinear',
      max_iter=500
   ))
])

# GridSearch: daftar kombinasi parameter yang akan diuji
params_grid_lr = [
   {
      'feat_select__k': np.arange(2, 10),
      'clf__penalty': ['l1', 'l2'],
```

```
    'clf__C': [0.01, 0.1, 1, 10],
  },
  {
    'feat_select': [SelectPercentile()],          # alternatif: seleksi berdasar persentase
    'feat_select__percentile': np.arange(20, 80, 10),
    'clf__penalty': ['l1', 'l2'],
    'clf__C': [0.01, 0.1, 1, 10],
  }
]

# Stratified K-Fold: menjaga proporsi label seimbang di setiap fold CV
SKF = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Jalankan GridSearchCV untuk mencari kombinasi parameter terbaik
gscv_lr = GridSearchCV(
  pipe_lr,
  params_grid_lr,
  cv=SKF,
  scoring='f1',     # metrik utama: F1-score
  verbose=1,        # tampilkan progres selama proses
  n_jobs=-1         # gunakan semua core CPU
)

print("Menjalankan GridSearch untuk Logistic Regression...")
start = time.time()
gscv_lr.fit(X_train, y_train)
print(f"GridSearch Logistic Regression selesai dalam {time.time() - start:.2f} detik")

# Tampilkan hasil terbaik dari GridSearch
print("CV Score (F1) terbaik:", gscv_lr.best_score_)
print("Kombinasi model terbaik:", gscv_lr.best_estimator_)
```

```
# Hitung akurasi model terbaik pada data uji
lr_test_score = gscv_lr.best_estimator_.score(X_test, y_test)
print("\nSkor Test (akurasi) Logistic Regression:", lr_test_score)

# Tampilkan fitur terbaik (jika feature selector mendukung metode get_support)
selector = gscv_lr.best_estimator_.named_steps['feat_select']
if hasattr(selector, 'get_support'):
    mask = selector.get_support()
    selected = np.array(X.columns)[mask]
    print("\nFitur terbaik (terpilih):", selected)

# Buat prediksi pada data uji dan tampilkan Confusion Matrix
lr_pred = gscv_lr.predict(X_test)
cm_lr = confusion_matrix(y_test, lr_pred)
disp_lr = ConfusionMatrixDisplay(confusion_matrix=cm_lr, display_labels=['0 = Hujan','1 = Cerah'])
disp_lr.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix — Logistic Regression")
plt.show()

# Tampilkan classification report (precision, recall, f1-score)
print("\nClassification Report — Logistic Regression:\n", classification_report(y_test, lr_pred))

# ============================================================
#  PIPELINE: Scaling → Feature Selection → Random Forest
# ============================================================

# Rancang pipeline: gabungkan scaling, seleksi fitur, dan model Random Forest
pipe_rf = Pipeline(steps=[
    ('scaling', MinMaxScaler()),
```

```
    ('feat_select', SelectKBest()),
    ('clf', RandomForestClassifier(
        class_weight='balanced',
        random_state=42,
        n_estimators=-1
    ))
])

# GridSearch: dua jenis seleksi fitur (KBest dan Percentile) dengan kombinasi parameter model
params_grid_rf = [
    # Kandidat 1: pakai SelectKBest
    {
        'feat_select__k': np.arange(5, 15),       # jumlah fitur terbaik yang diuji
        'clf__n_estimators': [200, 300, 500],     # jumlah pohon
        'clf__max_depth': [None, 5, 10],          # batas kedalaman tiap pohon
        'clf__min_samples_split': [2, 5, 10]      # jumlah minimal sampel untuk split node
    },
    # Kandidat 2: pakai SelectPercentile
    {
        'feat_select': [SelectPercentile()],
        'feat_select__percentile': np.arange(30, 80, 10),
        'clf__n_estimators': [200, 300, 500],
        'clf__max_depth': [None, 5, 10],
        'clf__min_samples_split': [2, 5, 10]
    }
]

# StratifiedKFold: memastikan proporsi kelas tetap sama di setiap fold CV
SKF = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Jalankan GridSearchCV: mencari kombinasi parameter terbaik dengan metrik F1
```

```
gscv_rf = GridSearchCV(
    pipe_rf,
    params_grid_rf,
    cv=SKF,
    scoring='f1',
    verbose=1,
    n_jobs=-1
)

print("Menjalankan GridSearch untuk Random Forest...")
start = time.time()

gscv_rf.fit(X_train, y_train)

print(f"GridSearch Random Forest selesai dalam {time.time() - start:.2f} detik")

#  Evaluasi hasil GridSearch
print("CV Score (F1) terbaik:", gscv_rf.best_score_)
print("Kombinasi model terbaik:", gscv_rf.best_estimator_)

rf_test_score = gscv_rf.best_estimator_.score(X_test, y_test)
print("\nSkor Test (akurasi) Random Forest:", rf_test_score)

#  Fitur terbaik (jika selector mendukung get_support)
selector = gscv_rf.best_estimator_.named_steps['feat_select']
if hasattr(selector, 'get_support'):
    mask = selector.get_support()
    selected = np.array(X.columns)[mask]
    print("\nFitur terbaik (terpilih):", selected)

#  Confusion Matrix & Classification Report
```

```
rf_pred = gscv_rf.predict(X_test)

cm_rf = confusion_matrix(y_test, rf_pred)

disp_rf = ConfusionMatrixDisplay(confusion_matrix=cm_rf, display_labels=['0 = Hujan','1 = Cerah'])

disp_rf.plot(cmap=plt.cm.Greens)

plt.title("Confusion Matrix — Random Forest")

plt.show()


print("\nClassification Report — Random Forest:\n", classification_report(y_test, rf_pred))


# Buat figure dengan 2 subplot berdampingan (1 baris, 3 kolom)

fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(16, 4))


# Plot Confusion Matrix untuk Logistic Regression

disp_lr.plot(ax=ax1, cmap=plt.cm.Blues, colorbar=False)

ax1.set_title("Logistic Regression (L1)")   # judul subplot pertama


# Plot Confusion Matrix untuk Random Forest

disp_rf.plot(ax=ax2, cmap=plt.cm.Greens, colorbar=False)

ax2.set_title("Random Forest (Best Model)") # judul subplot ketiga


# Rapikan tata letak agar subplot tidak tumpang tindih

plt.tight_layout()

plt.show()  # tampilkan semua plot


import pickle


# Simpan pipeline terbaik (bukan hanya model clf)

best_rf_pipeline = gscv_rf.best_estimator_


# Simpan pipeline lengkap ke file pickle
```

```
with open("BestModel_CLF_RandomForest_pingouin.pkl", "wb") as f:
    pickle.dump(best_rf_pipeline, f)

print("✅ Pipeline Random Forest terbaik berhasil disimpan ke 'BestModel_CLF_RandomForest_pingouin.pkl'")
```

Model GCS dan SVM

```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedKFold, GridSearchCV
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.preprocessing import MinMaxScaler,  StandardScaler
from sklearn.feature_selection import SelectKBest, SelectPercentile, mutual_info_classif
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier

from sklearn.pipeline import Pipeline
from sklearn.base import clone
import time

from sklearn.metrics import (
    confusion_matrix,
    ConfusionMatrixDisplay,
    classification_report
)
```

```
from sklearn.metrics import (
   confusion_matrix,
   ConfusionMatrixDisplay,
   classification_report
)

df_cuaca = pd.read_csv('dpc.csv',header=0)

df_cuaca.head()
print("Jumlah baris, kolom:", df_cuaca.shape)
print("\nTipe data:")
print(df_cuaca.dtypes)
df_cuaca = df_cuaca.drop(columns=['Tekanan (hPa)', 'pH Tanah', 'Kelembapan Tanah (%)'], errors='ignore')

# Tampilkan 5 baris pertama untuk verifikasi
df_cuaca.head()
# (1) Cek nilai NULL
print("\nJumlah nilai kosong (null) per kolom:\n", df_cuaca.isnull().sum())

# (2) Cek nilai NaN
nan_counts = df_cuaca.isna().sum()
print("\nJumlah nilai NaN per kolom:\n", nan_counts)

median_suhu = df_cuaca['Suhu (°C)'].median()
median_awan = df_cuaca['Awan (%)'].median()

# Isi nilai NaN dengan median
df_cuaca['Suhu (°C)'] = df_cuaca['Suhu (°C)'].fillna(median_suhu)
```

```
df_cuaca['Awan (%)'] = df_cuaca['Awan (%)'].fillna(median_awan)
df_cuaca['Kelembapan (%)'] = df_cuaca['Kelembapan (%)'].fillna(median_awan)
df_cuaca['Curah Hujan (mm)'] = df_cuaca['Curah Hujan (mm)'].fillna(median_awan)

# Tampilkan hasil median
print("Median Suhu (°C):", median_suhu)
print("Median Awan (%):", median_awan)

# Cek kembali apakah masih ada NaN
print("\nJumlah nilai kosong setelah penanganan:")
print(df_cuaca.isnull().sum())
# Validasi ulang
print("\nSetelah imputasi, nilai kosong per kolom:\n", df_cuaca.isnull().sum())
before = df_cuaca.shape
dupes = df_cuaca[df_cuaca.duplicated(keep=False)]
print(f"Jumlah baris duplikat (terhitung ganda): {dupes.shape[0]}")
df_cuaca2 = df_cuaca.drop_duplicates(keep='first')
print("Bentuk data sebelum/sesudah hapus duplikat:", before, "->", df_cuaca.shape)


# Deteksi OUTLIER dengan metode IQR
df_cuaca3 = df_cuaca2.select_dtypes(include=[np.number])
Q1 = df_cuaca3.quantile(0.25)
Q3 = df_cuaca3.quantile(0.75)
IQR = Q3 - Q1
outliers = ((df_cuaca3 < (Q1 - 1.5 * IQR)) | (df_cuaca3 > (Q3 + 1.5 * IQR))).any(axis=1)
print("\nJumlah baris yang terdeteksi memiliki outlier:", outliers.sum())

# Hapus baris yang memiliki outlier
df_cuaca_clean = df_cuaca2.loc[~outliers]
print("Ukuran Data Asli:", df_cuaca2.shape)
```

```
print("Ukuran Data Setelah Outlier Dihapus:", df_cuaca_clean.shape)
print("Jumlah baris yang dihapus:", outliers.sum())
# ==============================
# Encode Label & Pisahkan Fitur–Target
# ==============================

# Hujan   → 0 (Cuaca Hujan)
# Cerah   → 1 (Cuaca Cerah)

# Mengubah label prakiraan cuaca dari huruf menjadi angka
df_cuaca2['Prakiraan Cuaca'] = df_cuaca2['Prakiraan Cuaca'].map({'Hujan': 0, 'Cerah': 1})

# Menentukan X sebagai fitur (semua kolom kecuali 'Prakiraan Cuaca')
X = df_cuaca2.drop(columns=['Prakiraan Cuaca'])

# Menentukan y sebagai target (kolom 'Prakiraan Cuaca')
y = df_cuaca2['Prakiraan Cuaca']

# Menampilkan 5 baris pertama dataframe setelah perubahan
df_cuaca2.head()

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state = 45, stratify=y
)
print("Ukuran X_train, X_test:", X_train.shape, X_test.shape)
# ============================================================
#  PIPELINE: Scaling → Feature Selection → Gradient Boosting
# ============================================================

pipe_gb = Pipeline(steps=[
    ('scaling', MinMaxScaler()),
```

```
    ('feat_select', SelectKBest()),
    ('clf', GradientBoostingClassifier(
        random_state=42
    ))
])

params_grid_gb = [
    # Kandidat 1: SelectKBest
    {
        'feat_select__k': np.arange(5, 15),
        'clf__n_estimators': [50, 100, 150],
        'clf__learning_rate': [0.001, 0.002, 0.01],
        'clf__max_depth': [1, 2, 3],
        'clf__min_samples_split': [10, 15, 20]
    },
]

# ------------------------------------------------------------
# 3 StratifiedKFold untuk menjaga proporsi kelas di setiap fold
# ------------------------------------------------------------
SKF = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# ------------------------------------------------------------
# 4 Jalankan GridSearchCV untuk mencari kombinasi terbaik
# ------------------------------------------------------------
gscv_gb = GridSearchCV(
    pipe_gb,
    params_grid_gb,
    cv=SKF,
    scoring='f1',        # gunakan F1 agar seimbang antara precision & recall
    verbose=1,
```

```
    n_jobs=-1
)

print("Menjalankan GridSearch untuk Gradient Boosting...")
start = time.time()

gscv_gb.fit(X_train, y_train)

print(f"GridSearch Gradient Boosting selesai dalam {time.time() - start:.2f} detik")
# ------------------------------------------------------------
# 5️⃣ Evaluasi hasil GridSearch
# ------------------------------------------------------------
print("CV Score (F1) terbaik:", gscv_gb.best_score_)
print("\nKombinasi model terbaik:\n", gscv_gb.best_estimator_)

gb_test_score = gscv_gb.best_estimator_.score(X_test, y_test)
print("\nSkor Test (akurasi) Gradient Boosting:", gb_test_score)

# ------------------------------------------------------------
# 6️⃣ Fitur terbaik (jika selector mendukung get_support)
# ------------------------------------------------------------
selector = gscv_gb.best_estimator_.named_steps['feat_select']
if hasattr(selector, 'get_support'):
    mask = selector.get_support()
    selected = np.array(X.columns)[mask]
    print("\nFitur terbaik (terpilih):", selected)

# ------------------------------------------------------------
# 7️⃣ Confusion Matrix & Classification Report
# ------------------------------------------------------------
gb_pred = gscv_gb.predict(X_test)
```

```
cm_gb = confusion_matrix(y_test, gb_pred)
disp_gb = ConfusionMatrixDisplay(confusion_matrix=cm_gb, display_labels=['0 = Hujan','1 = Cerah'])
disp_gb.plot(cmap=plt.cm.Oranges)
plt.title("Confusion Matrix — Gradient Boosting")
plt.show()

print("\nClassification Report — Gradient Boosting:\n", classification_report(y_test, gb_pred))
pipe_svm = Pipeline(steps=[
  ('scaling', MinMaxScaler()),
  ('feat_select', SelectKBest()),
  ('clf', SVC(
    class_weight='balanced',   # menyeimbangkan kelas minoritas
    probability=True,        # agar bisa menghitung probabilitas prediksi
    random_state=42
  ))
])

# ------------------------------------------------------------
# 2 GridSearch: kombinasi parameter seleksi fitur & model
# ------------------------------------------------------------
params_grid_svm = [
  # Kandidat 1: SelectKBest
  {
    'feat_select__k': np.arange(5, 15),
    'clf__C': [0.1, 1, 10, 100],          # regularisasi
    'clf__kernel': ['linear', 'rbf', 'poly'],# jenis kernel
    'clf__gamma': ['scale', 'auto']        # parameter kernel RBF/poly
  },
  # Kandidat 2: SelectPercentile
```

```
  {
    'feat_select': [SelectPercentile()],
    'feat_select__percentile': np.arange(30, 80, 10),
    'clf__C': [0.1, 1, 10, 100],
    'clf__kernel': ['linear', 'rbf', 'poly'],
    'clf__gamma': ['scale', 'auto']
  }
]

# ------------------------------------------------------------
# 3️⃣ StratifiedKFold untuk validasi berstrata
# ------------------------------------------------------------
SKF = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# ------------------------------------------------------------
# 4️⃣ Jalankan GridSearchCV
# ------------------------------------------------------------
gscv_svm = GridSearchCV(
  pipe_svm,
  params_grid_svm,
  cv=SKF,
  scoring='f1',
  verbose=1,
  n_jobs=-1
)

print("Menjalankan GridSearch untuk Support Vector Machine...")
start = time.time()

gscv_svm.fit(X_train, y_train)
```

```
print(f"GridSearch SVM selesai dalam {time.time() - start:.2f} detik")

# ------------------------------------------------------------
# 5️⃣ Evaluasi hasil GridSearch
# ------------------------------------------------------------
print("CV Score (F1) terbaik:", gscv_svm.best_score_)
print("\nKombinasi model terbaik:\n", gscv_svm.best_estimator_)

svm_test_score = gscv_svm.best_estimator_.score(X_test, y_test)
print("\nSkor Test (akurasi) SVM:", svm_test_score)

# ------------------------------------------------------------
# 6️⃣ Fitur terbaik (jika selector mendukung get_support)
# ------------------------------------------------------------
selector = gscv_svm.best_estimator_.named_steps['feat_select']
if hasattr(selector, 'get_support'):
    mask = selector.get_support()
    selected = np.array(X.columns)[mask]
    print("\nFitur terbaik (terpilih):", selected)

# ------------------------------------------------------------
# 7️⃣ Confusion Matrix & Classification Report
# ------------------------------------------------------------
svm_pred = gscv_svm.predict(X_test)

cm_svm = confusion_matrix(y_test, svm_pred)
disp_svm = ConfusionMatrixDisplay(confusion_matrix=cm_svm, display_labels=['0 = Hujan','1 = Cerah'])
disp_svm.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix — Support Vector Machine")
plt.show()
```

```
print("\nClassification Report — Support Vector Machine:\n", classification_report(y_test, svm_pred))

# Buat figure dengan 2 subplot berdampingan (1 baris, 3 kolom)

fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(16, 4))


# Plot Confusion Matrix untuk Logistic Regression

disp_gb.plot(ax=ax1, cmap=plt.cm.Oranges, colorbar=False)

ax1.set_title("Gradient Boosting Classifier")   # judul subplot pertama


# Plot Confusion Matrix untuk Random Forest

disp_svm.plot(ax=ax2, cmap=plt.cm.Blues, colorbar=False)

ax2.set_title("Support Vector Machine") # judul subplot ketiga


# Rapikan tata letak agar subplot tidak tumpang tindih

plt.tight_layout()

plt.show()  # tampilkan semua plot


import pickle


# Ambil model terbaik dari hasil GridSearchCV

best_gb_model = gscv_gb.best_estimator_.named_steps['clf']


# Simpan model terbaik ke file pickle

with open("BestModel_CLF_GradientBoost_pingouin.pkl", "wb") as f:

   pickle.dump(best_gb_model, f)


print("✅ Model Gradient Boosting terbaik berhasil disimpan ke 'BestModel_CLF_GradientBoost_pingouin.pkl'")
```

python Streamlit

```
import streamlit as st
import pandas as pd
import numpy as np
import joblib
import altair as alt
from pathlib import Path

from streamlit_option_menu import option_menu

# Navigasi sidebar
with st.sidebar:
    selected = option_menu("Tugas UTS Streamlit UTS ML 24/25",
                           ['Upload Dataset',
                            'Klasifikasi Cuaca',
                            'Catatan'],
                           default_index=0)

if selected == 'Upload Dataset':
    st.header("")

CSV_PATH = "dpc.csv"
MODEL_PATH = "BestModel_CLF_RandomForest_pingouin.pkl"
UI_FEATURES = ["Suhu (°C)", "Kelembapan (%)"]
LABEL_MAP = {0: "Hujan", 1: "Cerah"}

st.set_page_config(page_title="Prediksi Cuaca", page_icon="🌤️", layout="centered")
st.title("🌦️ Prediksi Cuaca (Klasifikasi)")

st.markdown("""
<style>
```

```
/* Warna dasar aplikasi */
.stApp {
  background: linear-gradient(180deg, #0a0a0a 0%, #1a1a1a 100%);
  color: #f0f0f0;
}

/* Panel transparan di atas background hitam */
.panel {
  padding: 18px;
  border-radius: 16px;
  background: rgba(255, 255, 255, 0.08);
  border: 1px solid rgba(255, 255, 255, 0.1);
  box-shadow: 0 10px 30px rgba(0,0,0,0.4);
}

/* Tombol oranye gradasi */
.stButton > button {
  background: linear-gradient(90deg,#ff9966 0%, #ff5e62 100%);
  border: none;
  color: white;
  padding: 10px 16px;
  border-radius: 12px;
  font-weight:700;
  box-shadow: 0 10px 20px rgba(255,94,98,.35);
}
.stButton > button:hover {
  filter: brightness(1.1);
}

/* Teks & Header */
.title {
```

```
  font-weight: 800;
  font-size: 32px;
  background: linear-gradient(90deg,#36d1dc 0%,#5b86e5 100%);
  -webkit-background-clip: text;
  -webkit-text-fill-color: transparent;
  margin-bottom: 6px;
}
.subtle {
  color: #ccc;
  margin-bottom: 8px;
}

/* Nama tim animasi warna */
.rainbow {
  display:inline-block;
  font-weight:800;
  padding:4px 10px;
  border-radius:10px;
  background:linear-gradient(90deg,#ff6a88,#ffcc70,#a1c4fd,#c2ffd8,#ff6a88);
  -webkit-background-clip:text;
  -webkit-text-fill-color:transparent;
  animation:hueCycle 6s linear infinite;
}
@keyframes hueCycle {0%{filter:hue-rotate(0)}100%{filter:hue-rotate(360deg)}}

/* Chip probabilitas */
.metric-chip {
  display:inline-block;
  padding:10px 16px;
  border-radius:999px;
  background:linear-gradient(90deg,#36d1dc 0%,#5b86e5 100%);
```

```
  color:#fff;
  font-weight:700;
  box-shadow:0 8px 20px rgba(91,134,229,.28);
  margin-top:8px;
}

/* Footer */
.footer{
  text-align:center;
  color:#aaa;
  font-size:13px;
  margin-top:12px;
}
</style>

<div class="wrap">
  <div class="subtle">
    Masukkan <b>Suhu (°C)</b> & <b>Kelembapan (%)</b>. Fitur lain dilengkapi otomatis dari median dataset.
  </div>
  <div class="subtle">Dibuat oleh <span class="rainbow">BENY, DENIS, RENALDI</span></div>
</div>
""", unsafe_allow_html=True)

if not Path(CSV_PATH).exists():
  st.error(f"File {CSV_PATH} tidak ditemukan di folder ini.")
  st.stop()
if not Path(MODEL_PATH).exists():
  st.error(f"File model {MODEL_PATH} tidak ditemukan di folder ini.")
  st.stop()
```

```
df = pd.read_csv(CSV_PATH)
model = joblib.load(MODEL_PATH)

st.write('Untuk Inputan File dataset (csv) bisa menggunakan st.file_uploader')
file = st.file_uploader("Masukkan File", type=["csv", "txt"])
if file is not None:
    try:
        df = pd.read_csv(file)
        st.success("File berhasil diunggah!")
        st.dataframe(df)  # menampilkan isi CSV di Streamlit
    except Exception as e:
        st.error(f"Terjadi kesalahan saat membaca file: {e}")

def ensure_features(X_input, model, df):
    expected = list(model.feature_names_in_)
    for f in expected:
        if f not in X_input.columns:
            if f in df.columns:
                X_input[f] = pd.to_numeric(df[f], errors="coerce").median()
            else:
                X_input[f] = 0
    return X_input[expected].apply(pd.to_numeric, errors="coerce")

def pretty_label(y_pred, model):
    if hasattr(model, "classes_"):
        if isinstance(y_pred, str):
            return y_pred
        return LABEL_MAP.get(int(y_pred), str(y_pred))
    return LABEL_MAP.get(int(y_pred), str(y_pred))

st.markdown('<div class="wrap"><div class="panel">', unsafe_allow_html=True)
```

```
if selected == 'Klasifikasi Cuaca':
   st.header("Input Fitur Cuaca")
c1, c2 = st.columns(2)
with c1:
   suhu = st.number_input("Suhu (°C)", value=float(pd.to_numeric(df["Suhu (°C)"], errors="coerce").median()))
with c2:
   kelembapan = st.number_input("Kelembapan (%)", value=float(pd.to_numeric(df["Kelembapan (%)"], errors="coerce").median()))
st.markdown('</div></div>', unsafe_allow_html=True)


st.markdown('<div class="wrap"><div class="panel">', unsafe_allow_html=True)
X_user = pd.DataFrame([[suhu, kelembapan]], columns=UI_FEATURES)
X = ensure_features(X_user, model, df)
if st.button("🔍 Prediksi Cuaca"):
   try:
      y_pred = model.predict(X)[0]
      hasil = pretty_label(y_pred, model)
      proba = model.predict_proba(X)[0] if hasattr(model, "predict_proba") else None
      st.success(f"Hasil Prediksi: **{hasil}**")
      if proba is not None and hasattr(model, "classes_"):
         kelas = [pretty_label(c, model) for c in model.classes_]
         prob_df = pd.DataFrame({"Kelas": kelas, "Probabilitas": proba})
         color_range = ["#7bdff2", "#f2b5d4", "#b2f7ef", "#f7d6e0", "#c5d6ff"][:len(prob_df)]
         chart = alt.Chart(prob_df).mark_bar(cornerRadiusTopLeft=10, cornerRadiusTopRight=10).encode(
            x=alt.X("Kelas:N", title="Kelas Cuaca"),
            y=alt.Y("Probabilitas:Q", title="Probabilitas", scale=alt.Scale(domain=[0,1])),
            color=alt.Color("Kelas:N", scale=alt.Scale(range=color_range), legend=None),
            tooltip=[alt.Tooltip("Kelas:N"), alt.Tooltip("Probabilitas:Q", format=".2%")]
         ).properties(height=280)
```

```
            st.altair_chart(chart, use_container_width=True)
            st.markdown(f'<div class="metric-chip">🔮 Probabilitas Prediksi: {proba[np.argmax(proba)]*100:.2f}%</div>', unsafe_allow_html=True)
        else:
            st.info("Model tidak menyediakan probabilitas (predict_proba).")
    except Exception as e:
        st.error(f"Gagal memprediksi: {e}")
st.markdown('</div></div>', unsafe_allow_html=True)


st.markdown('<div class="wrap"><div class="footer">Made with ❤ • Beny Denis Renaldi</div></div>', unsafe_allow_html=True)


if selected == '📝 Catatan':
    st.header("📝 Catatan Penting – Aplikasi Prediksi Cuaca")


    st.markdown("""
    ### 🧠 **1. Tujuan Aplikasi**
    Aplikasi ini digunakan untuk **memprediksi kondisi cuaca (Hujan atau Cerah)** berdasarkan **parameter suhu (°C)** dan **kelembapan (%)** menggunakan model klasifikasi **Random Forest** yang telah dilatih sebelumnya.

    Tujuannya adalah memberikan simulasi **penerapan Machine Learning (ML)** di bidang **analisis data cuaca** dalam bentuk **web interaktif berbasis Streamlit**.

    ### 📈 **2. Fitur Input & Output**

    **Input:**

    - Suhu (°C)

    - Kelembapan (%)

    - Dataset eksternal (opsional)


    **Output:**

    - Prediksi kondisi cuaca: 🌧️ *Hujan* atau 🌞 *Cerah*

    - Probabilitas hasil prediksi dalam bentuk grafik (Altair).

    - Nilai probabilitas tertinggi ditampilkan dalam *metric chip* berwarna.
```

""")