LAMPIRAN

Kode Program :

```
import pandas as pd
import numpy as np
from datetime import datetime
from sklearn.model_selection import train_test_split, GridSearchCV,
cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVR
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.inspection import permutation_importance
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats

def load_data(path_excel):
    df = pd.read_excel(path_excel)
    df['AYAM DOC MASUK'] = pd.to_datetime(df['AYAM DOC MASUK'],
dayfirst=True, errors='coerce')
    df['AYAM PANEN'] = pd.to_datetime(df['AYAM PANEN'], dayfirst=True,
errors='coerce')
    df['MASA_PEMELIHARAAN'] = (df['AYAM PANEN'] - df['AYAM DOC
MASUK']).dt.days
    df['BERAT_RATA_EKOR'] = df['TOTAL DAGING'] / df['JUMLAH AYAM
PANEN']
    df['TINGKAT_KEMATIAN'] = (df['JUMLAH AYAM MASUK'] -
df['JUMLAH AYAM PANEN']) / df['JUMLAH AYAM MASUK']
    df['FCR'] = df['TOTAL PAKAN YANG DI KONSUMSI'] / df['TOTAL
DAGING']
    df = df[(df['JUMLAH AYAM MASUK'] >= 3000) & (df['JUMLAH AYAM
MASUK'] <= 18000)]
    df = df[(df['BERAT_RATA_EKOR'] >= 1.5) & (df['BERAT_RATA_EKOR']
<= 2.9)]
    df_clean = df.dropna(subset=['TOTAL DAGING',
'MASA_PEMELIHARAAN', 'JUMLAH AYAM MASUK',
'TINGKAT_KEMATIAN', 'FCR'])
    return df_clean

def analyze_feature_importance(model, X, y, feature_names):
    perm_importance = permutation_importance(model, X, y, n_repeats=10,
random_state=42)
    importance_df = pd.DataFrame({'Feature': feature_names, 'Importance':
perm_importance.importances_mean})
    importance_df = importance_df.sort_values('Importance', ascending=False)
    plt.figure(figsize=(10, 6))
    sns.barplot(x='Importance', y='Feature', data=importance_df)
```

```
    plt.title('Pentingnya Fitur (Permutation Importance)')
    plt.show()

    X_df = pd.DataFrame(X, columns=feature_names)
    X_df['Target'] = y
    correlation_matrix = X_df.corr()
    plt.figure(figsize=(10, 8))
    sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', center=0)
    plt.title('Matriks Korelasi Fitur')
    plt.show()

    return importance_df

def perform_cross_validation(model, X, y, cv=5):
    r2_scores = cross_val_score(model, X, y, cv=cv, scoring='r2')
    mae_scores = -cross_val_score(model, X, y, cv=cv,
scoring='neg_mean_absolute_error')
    rmse_scores = np.sqrt(-cross_val_score(model, X, y, cv=cv,
scoring='neg_mean_squared_error'))

    print("\n=== Hasil Validasi Silang ===")
    print(f"R² mean: {r2_scores.mean():.3f} (+/- {r2_scores.std() * 2:.3f})")
    print(f"MAE mean: {mae_scores.mean():.2f} (+/- {mae_scores.std() * 2:.2f})")
    print(f"RMSE mean: {rmse_scores.mean():.2f} (+/- {rmse_scores.std() *
2:.2f})")

    plt.figure(figsize=(12, 4))
    plt.subplot(131)
    plt.boxplot(r2_scores)
    plt.title('Distribusi R²')

    plt.subplot(132)
    plt.boxplot(mae_scores)
    plt.title('Distribusi MAE')

    plt.subplot(133)
    plt.boxplot(rmse_scores)
    plt.title('Distribusi RMSE')

    plt.tight_layout()
    plt.show()

def visualize_results(y_test, y_pred, df):
    plt.figure(figsize=(10, 6))
    plt.scatter(y_test, y_pred, alpha=0.5)
    plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', lw=2)
```

```
    plt.xlabel('Jumlah DOC Aktual')
    plt.ylabel('Jumlah DOC Prediksi')
    plt.title('Perbandingan Prediksi vs Aktual')
    plt.grid(True)
    plt.show()

    residuals = y_test - y_pred
    plt.figure(figsize=(10, 6))
    sns.histplot(residuals, kde=True)
    plt.xlabel('Residual')
    plt.ylabel('Frekuensi')
    plt.title('Distribusi Residual')
    plt.show()

    plt.figure(figsize=(10, 6))
    stats.probplot(residuals, dist="norm", plot=plt)
    plt.title('Q-Q Plot Residual')
    plt.show()

    plt.figure(figsize=(10, 6))
    sns.scatterplot(x=df['TOTAL DAGING'], y=df['JUMLAH AYAM MASUK'])
    plt.xlabel('Total Daging')
    plt.ylabel('Jumlah DOC')
    plt.title('Hubungan antara Total Daging dan Jumlah DOC')
    plt.show()

def train_model_daging_ke_DOC(df):
    feature_columns = ['TOTAL DAGING', 'MASA_PEMELIHARAAN', 'FCR',
'TINGKAT_KEMATIAN']
    X = df[feature_columns].copy()
    y = df['JUMLAH AYAM MASUK'].values

    print("\n=== Statistik Deskriptif Fitur ===")
    print(X.describe())

    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)
    X_scaled = pd.DataFrame(X_scaled, columns=feature_columns)

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

    param_grid = {
        'C': [0.1, 1, 10, 100],
        'epsilon': [0.01, 0.1, 0.5],
        'kernel': ['linear', 'rbf'],
```

```
        'gamma': ['scale', 'auto']
    }

    svr = SVR()
    gs = GridSearchCV(svr, param_grid, cv=5, scoring='neg_mean_squared_error',
n_jobs=-1)
    gs.fit(X_train, y_train)

    best_model = gs.best_estimator_
    print("\n=== Parameter Model Terbaik ===")
    print("Best Params:", gs.best_params_)

    y_pred = best_model.predict(X_test)
    print("\n=== Evaluasi pada Test Set ===")
    print(f"MAE: {mean_absolute_error(y_test, y_pred):.2f}")
    print(f"RMSE: {np.sqrt(mean_squared_error(y_test, y_pred)):.2f}")
    print(f"R²: {r2_score(y_test, y_pred):.3f}")

    perform_cross_validation(best_model, X_scaled, y)
    importance_df = analyze_feature_importance(best_model, X_scaled, y,
feature_columns)
    visualize_results(y_test, y_pred, df)

    return best_model, scaler, feature_columns

def predict_doc(model, scaler, feature_columns, **kwargs):
    """
    Fungsi prediksi dengan perbaikan penanganan nama kolom
    """
    try:
        # Konversi nama kolom ke format yang diharapkan model
        input_data = {
            'TOTAL DAGING': kwargs.get('total_daging', 0),
            'MASA_PEMELIHARAAN': kwargs.get('masa_pemeliharaan', 35),
            'FCR': kwargs.get('fcr', 1.6),
            'TINGKAT_KEMATIAN': kwargs.get('tingkat_kematian', 0.03)
        }

        # Buat DataFrame input
        df_input = pd.DataFrame([input_data])

        # Pastikan urutan kolom sesuai dengan feature_columns
        df_input = df_input[feature_columns]

        # Scaling input
        input_scaled = scaler.transform(df_input)
```

```
        # Prediksi
        doc_pred = model.predict(input_scaled)[0]

        # Batasi hasil prediksi
        doc_pred = max(3000, min(doc_pred, 18000))

        return doc_pred

    except Exception as e:
        raise ValueError(f"Error dalam prediksi: {str(e)}")

def run_main():
    try:
        df = load_data("Cleaned_Chicken_Dataset_For_Local.xlsx")
        print("Jumlah data setelah filter:", df.shape)

        model, scaler, feature_columns = train_model_daging_ke_DOC(df)

        nama_kandang = input("Masukkan nama kandang: ")
        total_daging = float(input("Masukkan berat total daging yang diinginkan
(kg): "))
        rencana_tgl_str = input("Masukkan rencana tanggal panen (DD MM
YYYY): ")
        rencana_tgl = pd.to_datetime(rencana_tgl_str, format="%d %m %Y")
        masa_pemeliharaan = 35
        fcr = 1.6
        tingkat_kematian = 0.03

        doc_pred = predict_doc(
            model,
            scaler,
            feature_columns,
            total_daging=total_daging,
            masa_pemeliharaan=masa_pemeliharaan,
            fcr=fcr,
            tingkat_kematian=tingkat_kematian
        )

        doc_pred_rounded = int(round(doc_pred))
        doc_date = rencana_tgl - pd.Timedelta(days=masa_pemeliharaan)

        print(f"\nRekomendasi untuk kandang {nama_kandang} menghasilkan
{total_daging:,.0f} kg daging")
        print(f"Membutuhkan {doc_pred_rounded:,} ekor DOC dengan berat rata-
rata {total_daging/doc_pred_rounded:.2f} kg/ekor.")
```

```
        print(f"Dokumentasi DOC masuk pada tanggal {doc_date.strftime('%d %B
%Y')}.")

    except Exception as e:
        print(f"\nTerjadi error: {str(e)}")

if __name__ == "__main__":
    run_main()
```